DECnet/E
Network Programming in MACRO-11

Order No. AA-L265A-TC

January 1982

This manual describes the DECnet/E services available to the MACRO-11 programmer, permitting him to design and implement programs that exchange data with other programs running in a DECnet network.

OPERATING SYSTEM AND VERSION:	**RSTS/E**	**V7.1**
SOFTWARE VERSION:	**DECnet/E**	**V2.0**

To order additional copies of this document, contact your local Digital Equipment Corporation Sales Office.

digital equipment corporation · maynard, massachusetts

First Printing, January 1982

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

310ALL

Contents

Figures

Tables

Preface

Manual Objectives

This manual describes the DECnet/E services available to a MACRO–11 programmer, permitting him to code programs that exchange data with other programs running in a DECnet network.

Intended Audience

This manual is written for the experienced programmer. While the reader is not expected to be knowledgeable of network programming, he is expected to be completely familiar with both the MACRO–11 programming language and the general mechanisms for interfacing with the RSTS/E operating system.

Prerequisite Reading

The reader is expected to be completely familiar with the concepts and features of the MACRO–11 programming language, as presented in the *PDP–11 MACRO Language Reference Manual*. The *RSTS/E Task Builder Manual* describes TKB, the linker used for building RSTS/E tasks from assembled MACRO–11 programs.

The reader is also expected to be familiar with the concepts and features available for interfacing to the RSTS/E system monitor, as presented in the following manuals:

RSTS/E System User's Guide
RSTS/E Programming Manual
RSTS/E System Directives Manual.

Related Documents

To obtain a general understanding of the DECnet environment — internal and external — the reader is directed to the *Introduction to DECnet*.

DECnet/E also provides utility programs that allow a terminal user to communicate with another terminal user, to manipulate files across the network, and to copy the contents of one storage device to a device on another system. The information needed to run and use these utilities is given in the companion manual, *DECnet/E Guide to User Utilities*.

Two other companion manuals tell how to generate and start a DECnet/E system (the *DECnet/E System Installation Guide*), control and manage a running DECnet/E system (the *DECnet/E System Manager's Guide*), and give other information useful to a system manager.

Three more DECnet/E manuals are available describing the DECnet/E message services for four other languages:

DECnet/E Network Programming in FORTRAN
DECnet/E Network Programming in BASIC-PLUS and BASIC-PLUS-2
DECnet/E Network Programming in COBOL

The more advanced network programmer may want to refer to the following two manuals for information concerning various network management functions:

DIGITAL Network Architecture, Network Services Functional Specication
DIGITAL Network Architecture, Network Management Functional Specication

Structure of This Manual

This manual is divided into three main parts. Part I, "Introduction," presents a general discussion of the DECnet/E environment (Chapter 1). Part II, "Background Concepts for Network Programming," presents a tutorial discussion of the DECnet/E features available for sending and receiving messages (Chapters 2 through 4). Part III, "Network Programming in MACRO," describes the MACRO call formats used to access those features (Chapter 5).

Part I
Introduction

Chapter 1
Introduction

DECnet/E is a combination of hardware and software that extends the capabilities of a RSTS/E system running on a DIGITAL PDP-11 computer to allow a programmer to develop and execute programs that exchange data with remote programs running on another DECnet system in a network.

NOTE

> Throughout this manual, the term *remote program* refers to any other program with which a program is communicating using the DECnet facilities. Such programs need not be located on another node in the network. It is possible to use DECnet/E to communicate with other programs at the local RSTS/E node. During development of network application programs, the whole system can be debugged locally and later distributed to remote nodes as required.

This manual describes the features available to network programmers in the MACRO-11 programming language on RSTS/E systems.

1.1 Basic Terms and Concepts

The term *network* is used here to refer to two or more computer systems connected such that they can exchange information. The computer systems, called *nodes*, are connected by communications paths called *physical links*. Physical links can be established through normal (switched) telephone circuits, several varieties of leased (private) lines, coaxial cables, or even satellite transmission facilities provided by several carriers.

In the network shown in Figure 1-1, a user at node BOSTON would refer to BOSTON as the *local node* and to nodes DENVER, DALLAS, LONDON, and LA as *remote nodes*. At node BOSTON, the nodes DENVER, DALLAS, and LONDON are physically *adjacent nodes*. That is, there is a direct physical link to these nodes with no intervening nodes. To a user at node LA, DENVER is the only adjacent node.

A *path* is the route over which data travels from its source to its destination within the network. *Path length* is the distance from the source node to the destination node, measured in hops. A *hop* is equal to a physical line between two adjacent nodes. In Figure 1-1, the path length between nodes DALLAS and BOSTON is one hop and between LA and LONDON is three hops. The *network diameter* is the maximum path length between any two nodes. The network shown in Figure 1-1 has a network diameter of three hops.

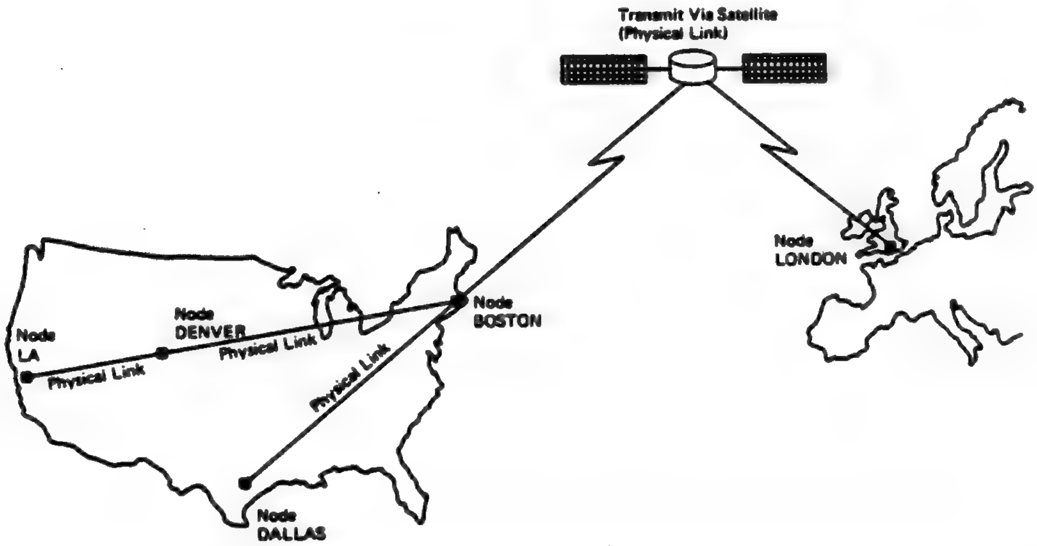


Figure 1-1: Computer Network Composed of Five Nodes

1.2 DECnet Networks

DECnet as a whole refers to the hardware and software that allows different DIGITAL systems to be connected to form networks. DECnet has been designed to allow each system to function as a separate entity but still allow access to resources that are distributed throughout the various systems in the network. A DECnet/E system provides all the capabilities of a normal RSTS/E system as well as the networking capabilities described in this manual and in the companion DECnet/E manuals listed in the Preface.

All DECnet implementations are based on a design structure called the DIGITAL Network Architecture (DNA). This architecture allows network implementations on the various DIGITAL systems to evolve (to provide more and more capability as time goes on) within a basic common structure. The structure ensures that implementations providing fewer features will always work with implementations providing more.

Perhaps a more immediate aspect of this process of evolution, however, is that at any given time capabilities can indeed differ from system to system. We do not attempt to describe here the DECnet implementations for all the different DIGITAL systems. Nevertheless, to use DECnet/E, you do need to give some thought to the other DIGITAL systems in your network.

If your network consists entirely of DECnet/E Version 2.0 nodes, this document provides all the information necessary for you to use DECnet/E to send and receive messages in MACRO-11 programs. If your network contains other DECnet systems, however, you should check the DECnet documentation for those systems to be sure that the corresponding capabilities exist. For example, you can design network programs to be started automatically on RSTS/E systems. This feature is not available on all DECnet systems, however, so your design should not necessarily assume this capability.

A brief overview of the functional capabilities of the various DECnet implementations can be found in *Introduction to DECnet*.

1.3 The DIGITAL Network Architecture (DNA)

DECnet/E is implemented according to a set of rules, or protocols, governing the format, control, and sequencing of data exchange from the user program level down through the physical link level. These protocols are defined by the DIGITAL Network Architecture (DNA) — a logical structure that provides the model for all DECnet implementations.

DNA consists of several layers, each defining a distinct set of network functions and a set of rules for implementing those functions. Each DECnet implementation consists of software modules that perform these DNA-defined functions according to DNA-defined protocols. The relevant protocols for network programming are called the Network Services Protocol (NSP), the Transport protocol, and the Digital Data Communications Message Protocol (DDCMP).

The Network Services Protocol defines the rules for communication between programs in a network over paths called logical links. Logical links, described in detail in Chapter 2, allow many different data streams to be multiplexed onto the physical link for transmission and separated at the receiving node for delivery to the appropriate program.

The Transport protocol defines the rules for determining the actual physical path, or *route*, along which data travels to its destination.

Physical link control is achieved in DECnet by implementation of the Digital Data Communications Message Protocol (DDCMP). DDCMP ensures an error-free, sequential data path over a generally error-prone medium. This is accomplished with the Cyclic Redundancy Check (CRC) for error detection, retransmission for error corrections, and numbered data segments to ensure sequential transmission of data.

1.4 Network Routing

DECnet/E Version 2.0 is a Phase III DECnet product. (Refer to *Introduction to DECnet* for a discussion of the differences between Phase II and Phase III DECnet products.) As such, it supports the adaptive routing feature of Phase III, providing the user with the capability of communicating with other nodes in the network regardless of whether or not they are directly connected to the local node.

Networks can consist of the following three types of nodes:

- **Routing nodes** — Phase III nodes connected to multiple communications lines, supporting route-through capabilities.
- **Nonrouting nodes (end nodes)** — Phase III nodes connected to a single communications line.
- **Phase II nodes** — nodes running a previous generation of the DECnet architecture.

Phase III nodes can communicate with and are compatible with Phase II nodes. However, Phase II nodes do not acquire any new capabilities by being connected to Phase III nodes, and the restriction that Phase II nodes can only communicate with adjacent nodes continues to apply. Thus, a Phase III node can communicate with any other Phase III node, *providing the path goes through Phase III nodes only.*

The adaptive routing feature is implemented through an algorithm that chooses the routing path with the lowest associated cost. Cost is computed as the sum of the costs of the lines over which the message is transmitted. The individual line cost parameters are assigned by the system manager and input into the system with the Network Control Program. (See the *DECnet/E System Manager's Guide* for a full discussion of line costs.) The algorithm automatically adjusts the routing when network topology or line cost changes occur.

1.5 Overview of DECnet/E Structure

The DECnet/E features for network communication are implemented as part of the RSTS/E system monitor. The manner in which these features are accessed differs with the type of user.

User programs interface with DECnet/E through NSP. Although it is not a separate entity or program, the term NSP is used throughout this manual to refer to the software that allows a user program to establish and exchange data over logical links. In other words, NSP is the implementation of the Network Services Protocol. The RSTS/E MACRO programmer accesses NSP services by coding calls to MACRO subroutines. NSP services can also be called from FORTRAN, BASIC-PLUS, BASIC-PLUS-2, or COBOL programs, as described in associated DECnet/E Network Programming Manuals.

Terminal users access certain network capabilities through various DECnet/E utility programs. The utilities TLK (Talk) and LSN (Listen) allow two terminal users to type messages to each other. The utilities NFT (Network File Transfer) and FAL (File Access Listener) work together to allow network file manipulation. They are implemented according to the Data Access Protocol (DAP) so that files can be exchanged with other DECnet systems having the same capabilities. The NET utility allows the terminal user to log into and use a remote RSTS/E system as though the local terminal were directly connected to the remote system, and NETCPY copies entire devices between RSTS/E nodes. (See the *DECnet/E Guide to User Utilities* for details on these utility programs.)

The system manager uses the features provided by the Network Control Program (NCP) to monitor and control a node running in a network. (See the *DECnet/E System Manager's Guide* for details on running NCP.)

Some elements of DECnet/E are not directly accessible to any user. These lower level elements of DECnet/E include the network Transport module (TRN) that implements the Transport protocol, several hardware devices (the DMC11, DMR11, DMV11, and DMP11), and their associated software device drivers.

The device drivers are responsible for handling messages received from and sent over the physical communication lines. NSP gives a message to be sent to Transport, which selects an appropriate data path based on the destination of the message. Transport then passes the message to the proper device driver for actual transmission. The drivers manipulate the device registers to cause message transmission to occur.

To handle incoming messages, the drivers include buffer management functions that allow messages to be received from the physical links. When a message is received, it is passed to Transport, which checks the destination of the message. If the message is for the local node, Transport passes it to NSP for further processing. If the message is destined for another node in the network, Transport selects an outgoing data path and passes the message to one of the drivers for transmission.

The hardware devices are intelligent communication controllers that connect the PDP-11 to physical communications lines. The DMC11 and the DMR11 control lines that connect to only one other system in the network (point-to-point lines) while the DMV11 and the DMP11 control lines that can connect to more than one other system (multipoint lines). The physical lines can include cables, modems and telephone circuits, or even satellite transmission facilities. Each controller contains a microprocessor, its own memory, and microcode that implements the DDCMP protocol. The implementation of DDCMP in firmware considerably reduces the software overhead for the DECnet/E system.

Part II
Background Concepts for Network Programming

Chapter 2
Logical Links

This chapter introduces the logical link — the basic element of DECnet task-to-task communication. A logical link is a software path for the exchange of data between two programs running in a network.

In a network of DECnet nodes, many programs use the same physical path to exchange data. Data being sent from one program to another over a logical link is interspersed (multiplexed) with data sent over other logical links, transmitted over the physical line, and separated (demultiplexed) at the receiving end. The Network Services Protocol (NSP), implemented as a part of the DECnet software at each node, handles this multiplexing and demultiplexing. It accepts outgoing data from local programs and formats it for transmission by the communications hardware. NSP also accepts incoming data from remote nodes and separates it into individual logical link streams which are then delivered to the appropriate local programs (Figure 2-1).

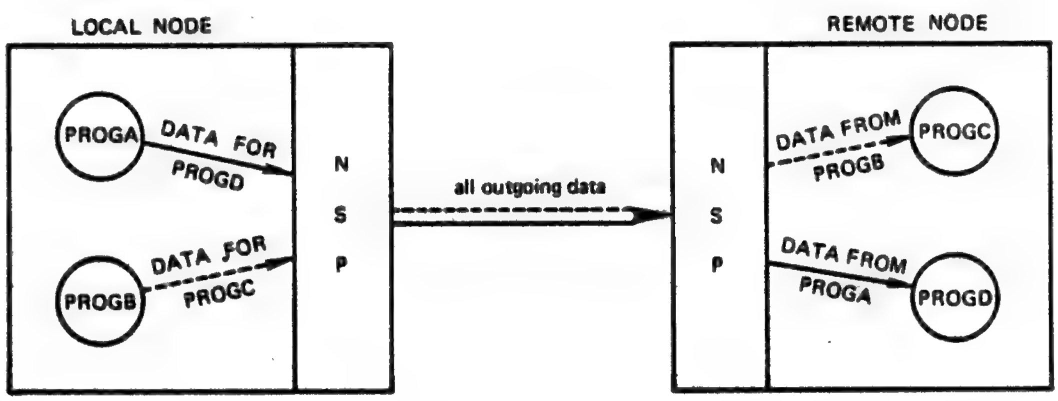


Figure 2-1: NSP Handles Multiplexing and Demultiplexing of Data over Physical Lines

Throughout the remainder of this chapter, and the two chapters that follow, various logical link parameters are discussed — object name and type code, local and remote link address, flow control data request counts, and so on. The parameters can be observed on an operational node, by issuing the NCP SHOW LINK command (for a single link) or the SHOW ACTIVE LINK (for all current links). This can be very helpful when debugging a network program. (See the *DECnet/E System Manager's Guide* for details on using these NCP commands.)

2.1 Creating a Logical Link

Creating a logical link is a cooperative venture. Two programs must agree to communicate before a logical link is established. The procedure is basically the same for all implementations of DECnet. The program that initiates the request for a logical link connection is called the source. The other program is called the target. This distinction is made only during the connection sequence.

NOTE

> Other network implementations make the source program the "link master." The source must poll the target for data and only the source can disconnect the link. DECnet does not make this distinction. Once the connection is established, the terms "source" and "target" have no significance and both the local and remote programs have equal access to the logical link.

The request for a connection takes the form of a Connect Initiate Message that includes detailed information identifying the remote node and target program to which the connection is to be made. It also establishes an identifier for the link itself. In DECnet/E, the link identifier is called a user link address, or ULA. A ULA is a number from 1 to 255 that is used by the source program to refer to the link during subsequent send and receive operations.

The NSP software at the source node does some initial checking of the request (to determine if it recognizes the remote node, for example). If all is well, the source NSP forwards the connect request to the NSP at the target node. The target NSP does its own checking to determine if the connection can be made. If the connection cannot be made, the target NSP rejects the connection. Otherwise, it delivers the connect request to the target program. The target program can then choose to accept or reject the logical link.

If the target program accepts the link, it selects its own link identifier by which it will refer to the link. The two link identifiers are unrelated to one another. They can even take quite different forms if the programs are written in different languages or for different operating systems. If the remote node is a DECnet/E node, this identifier is again called a user link address and can

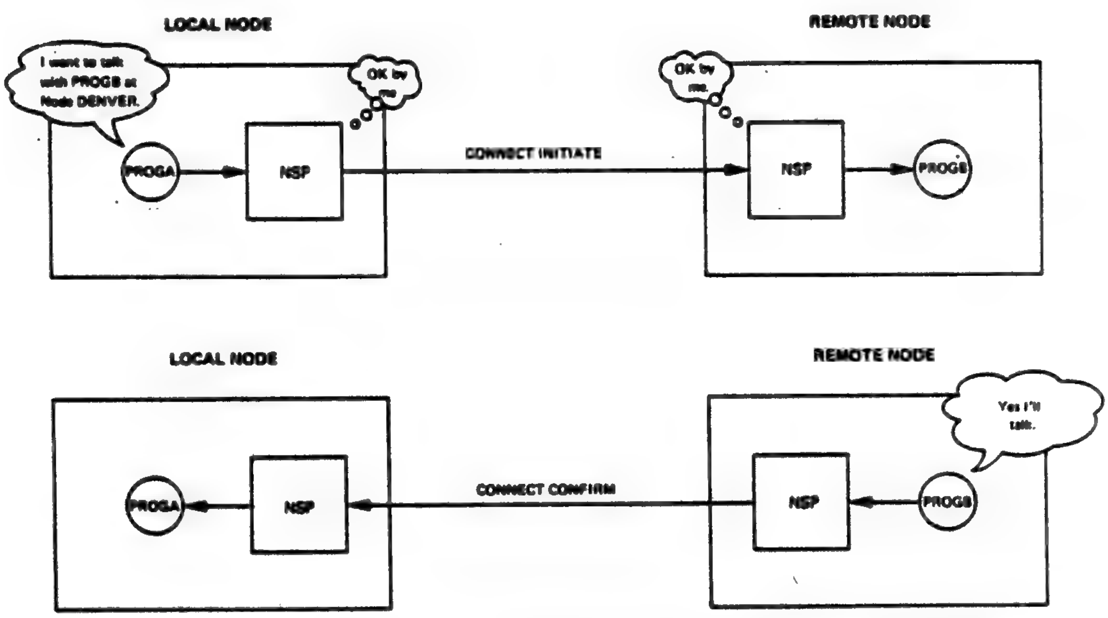
range from 1 to 255. The number selected is completely independent of the selection made by the source program. The source and target NSPs relate the two identifiers to the same logical connection and keep the data exchanged over the link separate from data exchanged over other logical links.

The target program's confirmation or rejection of the link is passed back over the network to the source NSP, which passes this information on to the source program. This confirmation or rejection takes the form of a Connect Confirm or Connect Reject Message that is queued for the source to receive and process.

If the connection is confirmed after this mutual handshaking, the two programs can then send and receive data over the link, each referring to the link by the identifier it assigned during the connection sequence. Both programs can send data simultaneously, since the DECnet software provides logical full duplex transmission regardless of the characteristics of the intervening network. Either program can break the logical link connection when it is no longer needed.

In Figure 2-2, A shows the procedure for establishing a logical link connection. The target NSP considers the link "up" (that is, it will allow the target program to send data) when the source NSP has acknowledged receipt of the Connect Confirm Message. The source NSP considers the link up when it receives the Connect Confirm Message. In Figures 2-2, B, C, and D illustrate the three ways in which a request for a link might be rejected.

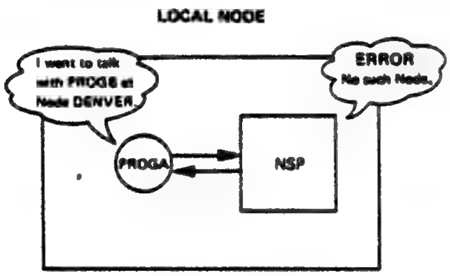
A. Remote program confirms logical link connection



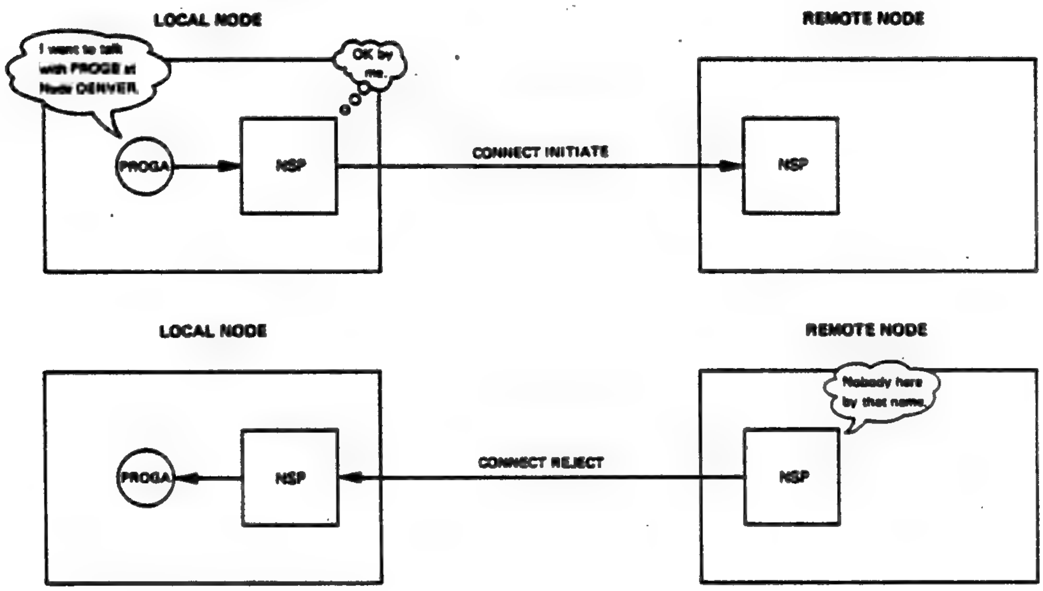
(continued on next page)

Figure 2-2: Logical Link Connections

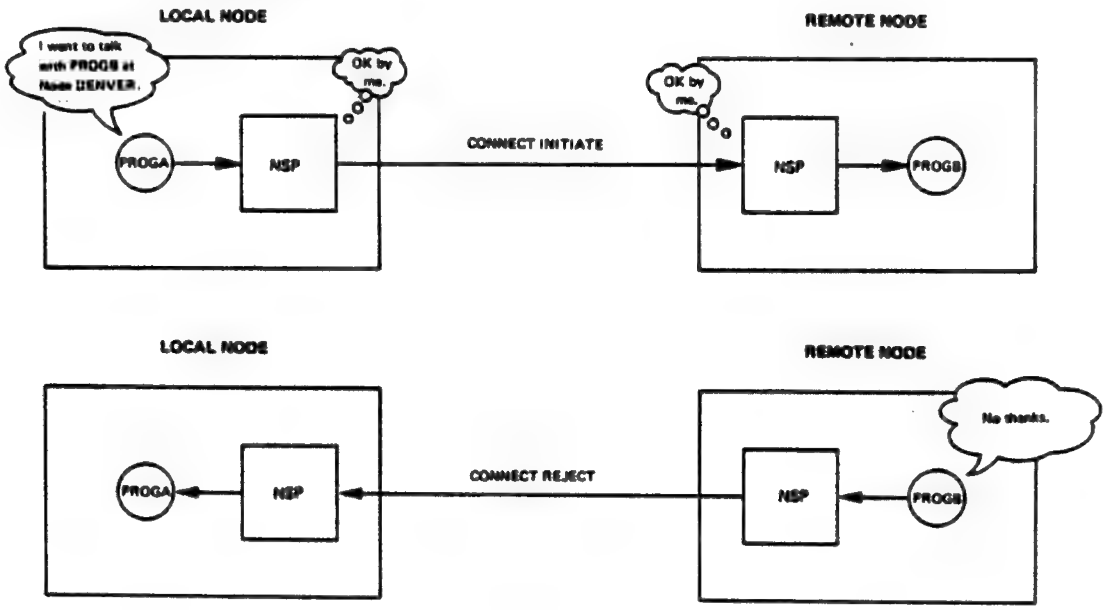
B. Local NSP returns an error on connect request



C. Remote NSP rejects logical link connection



D. Remote program rejects logical link connection



2.2 Advantages of Logical Links

In addition to making it possible to sort out the many physical streams of data that pass over a single physical line, the logical link structure maintained by NSP has several other advantages.

- Reduction of addressing data required for message transmission, thereby providing efficient line usage
- Separation of data into separate message streams
- Provision for high-priority "Interrupt" messages
- Ability to control the flow of data, thereby relieving possible network congestion

2.2.1 Efficient Line Usage

Once established, a logical link simplifies the addressing necessary for the exchange of data between two programs, both at the user's level and in transmission of data across the network. As mentioned previously, a link identifier is assigned by each program to identify an established logical link. This identifier is more convenient than the longer, detailed address used in identifying the remote node and program in the original Connect Initiate Message.

During a connection sequence, the source NSP and target NSP also establish a set of addresses that they use to identify the logical link. These addresses are called the local link address (LLA) and the remote link address (RLA), respectively. Each NSP refers to its own as the local link address and the other as the remote link address.

While individual ULAs must be unique to the source program and to the target program, they are by no means required to be unique throughout the network, or even within the local node. Thus, the LLA and the RLA are required by NSP to separate the many distinct message streams that are multiplexed onto the communications lines. Since each NSP's LLA is unique to its own node, these NSP addresses reduce the amount of control and identifying information that must accompany the user data transferred across the link (Figure 2-3).

Figure 2-4 illustrates how these addresses work, showing two DECnet/E nodes with two established logical links. Program A, for example, has established a logical link with Program D. Program A refers to the link with a user link address of 1. Program D refers to the link with a user link address of 23. Each NSP has established its own local link address. For Program A's NSP, the LLA is 123456. For Program D's NSP, the LLA is 024602. Each NSP keeps track of the other's LLA as a remote link address.

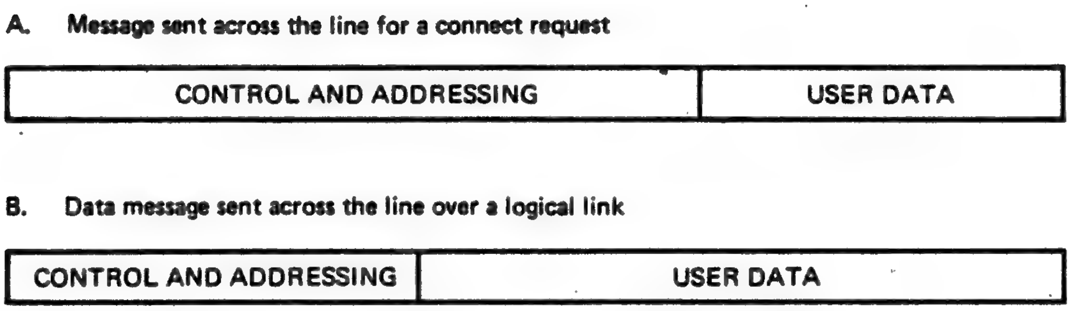


Figure 2-3: Logical Links Provide Line Efficiency

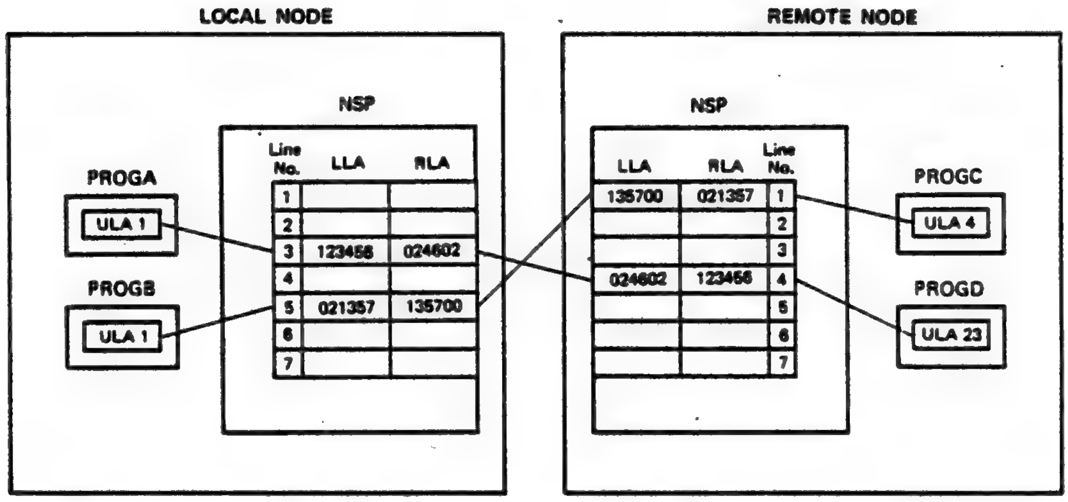


Figure 2-4: User Link Addresses, Remote Link Addresses, and Local Link Addresses for Logical Links

2.2.2 Separate Data by Intended Use

Logical links provide a means by which incoming messages can be separated into different message streams. A program can establish different logical links to communicate with different remote programs. Or it can establish several logical links with the same remote program to exchange data intended for different purposes.

Before a program can use the message services, it must first register this intention with NSP. This involves declaring the name by which the program is to be known to other remote programs in the network (that is, its identity), as well as any limits or restrictions that will affect the way in which the program can use the message services.

When a program declares its intent to use the network message services, NSP sets up a receive queue to hold pending messages. Thereafter, all messages received for the program are placed in the queue. Messages from different

logical links are interspersed in the queue but the receiving program can retrieve either the first message in the queue or the first message from a particular logical link.

The concept of separate message streams for separate logical links is illustrated in Figure 2-5.

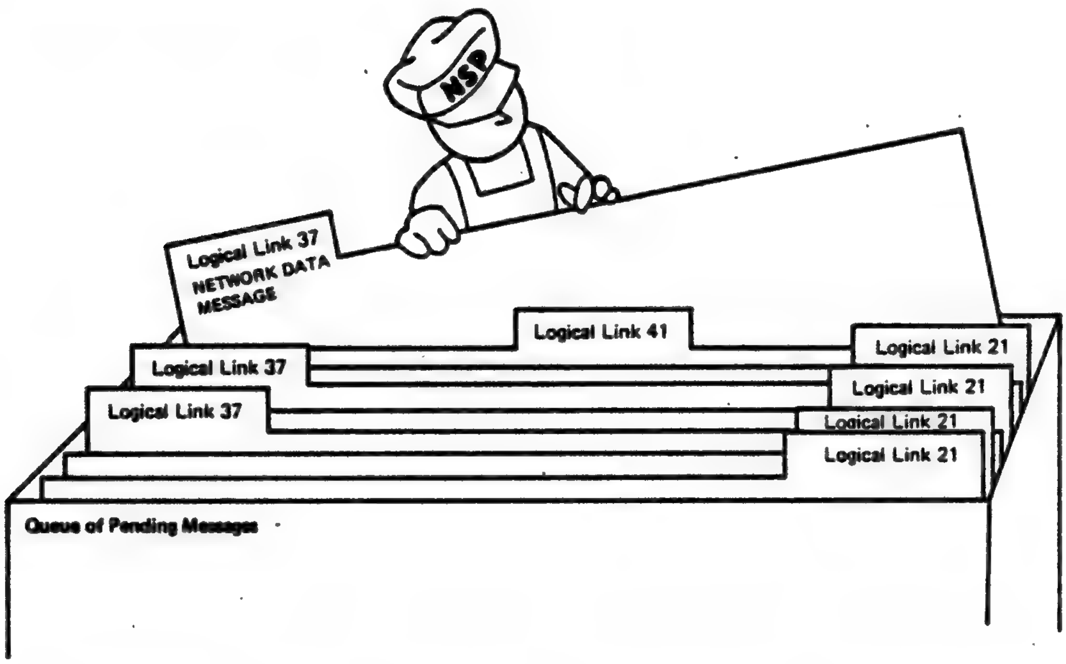


Figure 2-5: Data Is Queued for Logical Links in Separate Streams

2.2.3 Interrupt Messages

Special high-priority Interrupt Messages can be sent over logical links in all DECnet implementations. DECnet/E queues incoming Interrupt Messages at the head of the pending message queue, behind the first message in the queue (since the user program could be in the process of retrieving the first message) and behind any other pending Interrupt Messages queued for the program.

In the DECnet/E implementation, these Interrupt Messages are not true interrupts in the sense of causing an immediate jump to an interrupt processing routine. (Some DECnet implementations *do* treat Interrupt Messages as true interrupts.) They are simply messages that can be identified as different from ordinary data messages and that are placed at the head of the pending message queue (Figure 2-6).

If a program is designed to send Interrupt Messages, the receiver at the other end of the link must be set up to process them. Interrupt Messages can contain a small amount of user data so that programs can be designed to recognize Interrupt Messages for different purposes.

Interrupt Messages are discussed in more detail in Chapters 3 and 4.

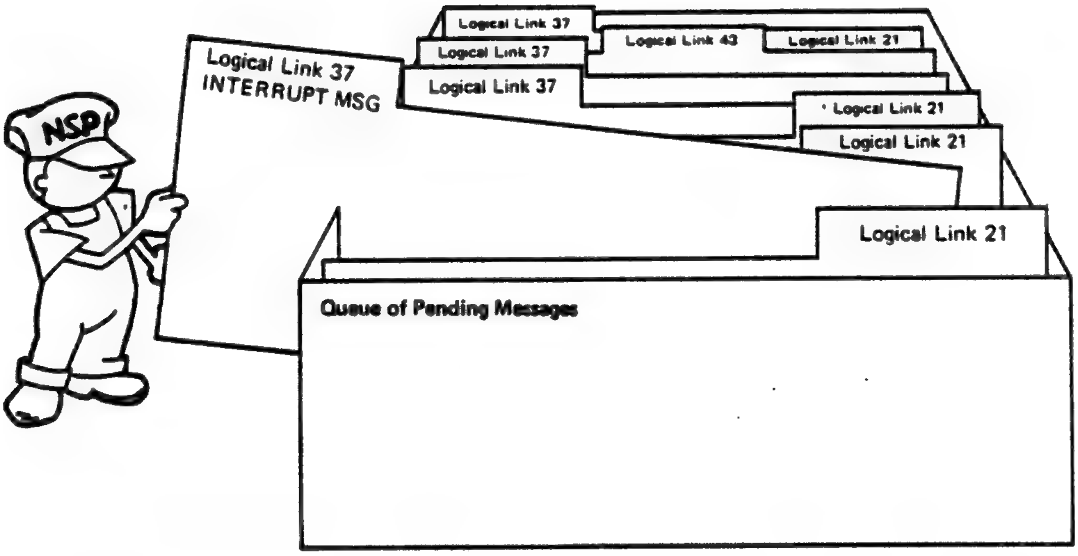


Figure 2-6: Queuing of Interrupt Messages for Logical Links

2.2.4 Flow Control

NSP maintains each program's queue of pending messages in system buffer space. A message is stored in the queue until it is received by the user program. Thus, pending messages tie up valuable system resources. Several flow control mechanisms are used in DECnet to protect systems from being flooded with messages. Some of these mechanisms are system-regulated while one mechanism is selected and controlled by the user program.

Each receiver program can select one of the following three types of program-regulated flow control:

- Segment flow control
- Message flow control
- No active flow control

A discussion of program-regulated flow control is given in Section 4.2.3.

2.3 DECnet/E Logical Link Limitations

The implementation of DECnet for RSTS/E allows a maximum of 127 active logical links for the entire node at any one time. The system manager can reduce this maximum to some number between 1 and 127. There is no limit to the number of logical links per program other than the system maximum.

Chapter 3
Overview of DECnet/E Message Services

RSTS/E provides message services for communication between local programs. The DECnet/E message services are implemented as an extension of these local services. In addition, DECnet/E offers new capabilities for local communication not found with the original RSTS/E message mechanism. The original local communication services are described in the *RSTS/E Programming Manual*. They are repeated in this document both for completeness and because they are the logical basis for the DECnet extensions.

We emphasize here that it is the way in which the user interfaces with DECnet from RSTS/E that is implemented as an extension of local capabilities. An RSX-11 programmer might interface with DECnet somewhat differently. The underlying architecture provides the consistency. The programmer at each system uses a familiar language and form. DECnet handles both the transmission itself and the translation necessary at both nodes to convert the original call to a form that can be transmitted over the line and back to the format expected at the receiving node.

Table 3-1 lists the calls used for interprogram communication by programs running under DECnet/E. The remaining sections of this chapter give a general description of each of the calls. Specific MACRO calls, with detailed formats, are given in Chapter 5.

Table 3-1: Summary of Calls for Interprogram Communication

Call	Function
Declare Receiver	Registers the program with the operating system for message services.
Remove Receiver	Terminates message operations for the program.
Get Local Node Parameters	Obtains local node information.
Log User Event	Queues a user-generated event to the local event processor for logging.
Send Local Data Message	Transmits user data to a local program.
Send Connect Initiate Message	Requests a logical link connection with another program.

(continued on next page)

Table 3–1 (Cont.): Summary of Calls for Interprogram Communication

Send Connect Confirm Message	Accepts a logical link connection requested by another program.
Send Connect Reject Message	Rejects a logical link connection requested by another program.
Send Network Data Message	Transmits user data to a network program over an established logical link.
Send Interrupt Message	Transmits interrupt data to a network program over an established logical link.
Send Link Service Message	Requests data over a flow controlled logical link.
Send Disconnect Message	Disconnects an established logical link, after all pending messages have been sent.
Send Link Abort Message	Disconnects an established logical link immediately, destroying any messages waiting to be sent.
Receive	Receives a message from the queue of pending messages. (One of nine message types, corresponding to the nine Send calls listed above, could be received.)

3.1 General Remarks

The Declare Receiver call informs the local RSTS/E monitor that the program issuing the call intends to communicate with other programs. This call identifies the program and defines which of the services will be required. The Remove Receiver call is issued when message operations are complete for the program and releases system resources used for message services. The Get Local Node Parameters call returns information to the calling program concerning the local node. The Log User Event call permits a user-written program to queue an event to the event processor for logging.

Beyond these, the operations available fall into two categories: send and receive. The entities sent and received are called messages. They consist of control information identifying the type of message, and in most cases, data of interest to the programs exchanging information. Nine message types are available for these operations.

A separate call is used to send each of the message types. That is, there is a Send Connect Initiate Message call, a Send Local Data Message call, and so forth. When one of these calls is executed, NSP uses the parameters and user data included in the call to format the appropriate network message into system buffer space. Necessary header information is also included. This message is then queued to Transport for transmission across the network. The way in which a message is queued for transmission depends on the type of message. (A discussion of transmission queuing is given in Section 4.2.1.)

The Receive call retrieves a message from the program's queue of pending messages maintained by NSP in system buffer space. NSP queues incoming messages for the program as they come in across the network. Messages from local senders are also placed in the pending message queue.

NOTE

> Within the current context, *local* programs are programs running at the local node that send messages using only the Send Local Data Message call. *Network* programs, on the other hand, are programs running either locally or remotely that use DECnet to establish and send data over logical links.

The Receive call returns control information identifying the sender and the type of message (one of the nine listed previously in Table 3–1). User data accompanying the message is returned to a user buffer specified in the Receive call where it can then be examined and processed.

A user program can issue general or selective Receive calls. On a general Receive, the first message in the queue is returned. A selective Receive can select the first message from any local sender, a particular local sender, any network sender, or a particular logical link.

There can be times when no pending messages are in the queue when a Receive call is issued. One option available in the Receive call causes the program to "sleep", or suspend execution, until some action takes place — until a message is queued or a specified amount of time has passed, for example. The Receive call can also be issued to retrieve only part of a message. A program with limited buffer space can choose to retrieve only part of the user data with one Receive call. Subsequent Receive calls will retrieve the rest of the data, or the remaining data can be discarded.

3.2 Registering with the Monitor: Declare Receiver Call

A program declares its intent to use the message services by issuing a Declare Receiver call. This call must be executed before a program can send network messages or receive messages of any type. A program can, however, send local data messages without executing this call.

The Declare Receiver call does more than just declare intent, however. It also defines the way in which other programs throughout the network are to address this program (the program's identity) and the way in which the program can thereafter use the message services.

The Declare Receiver call has some rather far-reaching effects. For this reason, the parameters specified with the call and how the system uses them are described here in detail.

3.2.1 Declaring Identity

A Declare Receiver call identifies the calling program with an object type code, a name, or both. The manner in which a program declares its identity affects the way incoming link connections are handled.

A name is an ASCII identifier that can be used by either local or network programs to refer to the declarer. No two programs at the local node can use the same name at the same time. If a name is given, it must be unique within the local node.

An object type code, on the other hand, is simply a number in the range of 1 to 255. It can be used by other network programs to refer to the declarer. (Local programs cannot address a receiver by object code.) These codes simplify the identification of programs performing identical functions at different nodes of a network, without going through the cumbersome process of arriving at and enforcing standard naming conventions.

An object code also need not be unique, and more than one receiver can declare identity with the same object code simultaneously. A program performing a general-purpose function of interest to the entire network can incur heavy use. By permitting more than one program to declare identity with the same object code at the same time, DECnet/E permits execution of multiple copies of a program to meet sporadic heavy usage.

A range of object type codes (1 through 127) is reserved for DECnet use. The remainder (128 through 255) are available for user applications programs. Currently assigned object type codes for DECnet use are given in Appendix A.

NOTE

> In the DECnet environment it is recommended that, whenever possible, programs declare their identity with and be addressed by nonzero object type codes.

Network addressing is discussed in detail in Chapter 4.

3.2.2 Declaring Intended Usage

In addition to identifying the program, the Declare Receiver call defines how the program intends to use the message services. For example, a program can limit the incoming messages to those from local senders only. Or, it can limit incoming messages to those from network senders only.

Other protective limits can be set in the Declare Receiver call.

- The maximum amount of system buffer space to be used for the pending message queue.
- The maximum number of messages that can be stored in the pending message queue.
- The maximum number of active logical links that the program can have at any given time.

These limits affect incoming messages and incoming requests for logical links. The Declare Receiver call does not limit the type or number of send calls a program can execute.

3.2.3 DECnet/E Use of the Declare Receiver Call

When a Declare Receiver call is issued, the system sets up a 32-byte Receiver ID Block (RIB) to associate the information with the RSTS/E job number. The Receiver ID Block actually applies to the job rather than the program

that issues the Declare Receiver call. This distinction is rather fine for most applications, but does make it possible, for example, to chain together a set of programs using message services. The first program in the series would execute a Declare Receiver call. Then other programs in the chain would also be valid receivers as long as they were always executed under the same job number.

The Receiver ID Block holds the arguments passed with the Declare Receiver call, along with other system- and NSP-maintained information, such as the address pointers of the pending message queue and the address of the list of active logical links.

3.3 Remove Receiver Call

A Declare Receiver call remains in effect until a Remove Receiver call is issued. The Remove Receiver call releases the Receiver ID Block. All pending messages are discarded and all active logical links are destroyed. (NSP sends a Link Abort Message to the remote programs for any active logical links.)

Like the Declare Receiver call, the Remove Receiver call applies to the RSTS/E job. If a program does not issue a Remove Receiver call before terminating execution, other programs executed later under this job number will be unable to issue a Declare Receiver call. (As far as the system is concerned, the old one is still in effect.) A Remove Receiver call should be issued as soon as message services are no longer needed. This practice prevents queuing of unwanted messages and releases the system buffer space used for the Receiver ID Block.

Also, any program that could be stopped by a terminal user's CTRL/C should have provisions to issue a Remove Receiver call and terminate gracefully. If a job is killed, NSP automatically destroys any active logical links, discards messages, and releases any system resources used for message operations.

NOTE

> It is generally good practice for a program to issue a Remove Receiver call immediately before issuing a Declare Receiver call. This prevents errors on the Declare Receiver call caused by abnormal termination of a previous program running under the same job number.

3.4 Get Local Node Parameters Call

The Get Local Node Parameters call returns information to the calling program concerning the local node. This information consists of the node name, address, and alias (if any), and the node's default project-programmer number (if any). (See the *DECnet/E System Manager's Guide* for a discussion of alias.)

A general-purpose network program — one that runs on several different nodes rather than being specific to any one node — would use this call to obtain the node name, number, and alias for use in the connection sequence.

The program uses this information to identify itself to the remote node and program. A program that is started automatically as the result of an incoming connect request would use this call to obtain the default project-programmer number for use in validating the connection and logging into a user account. (See "Automatic Job Startup" in Chapter 4 for a discussion of the default account.)

3.5 Log User Event Call

The Log User Event system call permits a user-written program to queue an event to the system event processor for logging. Before the event is logged, it is time-stamped by the system in the standard Network Information and Control Exchange (NICE) protocol format.

Optional parameter data can be passed to the event processor for processing and output. If it is included, this data must be in NICE protocol format. (See the *DIGITAL Network Architecture, Network Management Functional Specification*, Version 3.0.0, for further information on event logging.)

NOTE

> This system call performs a highly specialized function which requires a great deal of special knowledge of DECnet and the DIGITAL Network Architecture. It is provided as a convenience for the sophisticated network user and is not intended for normal network programs.

3.6 Send/Receive Local Data Message

The Send Local Data Message call transfers up to 532 bytes of user message data to a local program. The message is placed in the appropriate receive queue of the target program. The target program can be identified either by job number or by the name given in its Declare Receiver call.

A Receive call that returns a Local Data Message returns control information identifying it as a Local Data Message from a specific local sender, and up to 532 bytes of user data.

The Local Data Message is not used for DECnet communication but is included here for reference. A user program can carry on local conversations and maintain one or more DECnet logical links at the same time.

3.7 Send/Receive Connect Initiate Message

The Send Connect Initiate Message call is used to request a logical link connection with a remote program.

The target node and program are identified by a 120-byte Connect Data Block passed as part of the call. The following information is specified with the call:

> The user link address (ULA) that the source program will use to refer to this link.

The type of receiver flow control desired by this program — that is, the type of flow control to be imposed on incoming data messages. (A full discussion of flow control is deferred to Chapter 4.)

A receive maximum. The maximum amount of data to be received by the program over this link in one Network Data Message. The size of the receive buffers allocated by NSP will also limit the amount of data that can be passed in a single Network Data Message. If the maximum buffer size set by Network Management for NSP is not large enough to handle the receive maximum as specified by the user program, NSP will alter the receive maximum to the smaller limit before forwarding the Connect Initiate Message. The local program is informed of this change (if any is made) by a field in the Connect Confirm Message from the remote program when it accepts the link.

A Connect Data Block identifying the target node and program to which the connection is requested. The target program can be identified by name or object type code. NSP inserts the local program identification, as specified in the source program's Declare Receiver call, into the Connect Data Block. This allows the remote program to determine who sent the connect request and decide whether to accept or reject the connection. The Connect Data Block can also contain data required by the remote program or system for access control. Such data might consist of a project-programmer number, password, and account number — or whatever is required by the remote program or system.

Up to 16 bytes of user data.

A Receive call that returns a Connect Initiate Message from the queue of pending messages returns control information identifying it as a Connect Initiate Message from a specific network program, and requirements imposed by the remote program. The information includes the following:

The local link address (LLA) assigned by the local NSP to refer to this logical link. The program will use the LLA to refer to the link in its responding Connect Confirm or Connect Reject Message. (Note that the LLA is only used to refer to a link while no user link address (ULA) is in effect. The ULA is established by the receiving program in its Connect Confirm Message that accepts the connect request and is used thereafter by the program to refer to the link.)

The type of receive flow control desired by the remote program — that is, the type of flow control that will be imposed on outgoing data messages over this link.

A receive maximum. The maximum amount of user data that can be received by the local program over the link. This maximum is calculated by the local NSP, based on the size of receive buffers it allocates. The program can reduce this maximum further in its responding Connect Confirm Message.

A transmit maximum. The maximum amount of user data that can be transmitted by the program over this logical link in one Network Data

Message. This maximum is imposed by either the remote NSP or the remote program and corresponds to the remote program's receive maximum.

A 120-byte Connect Data Block, identifying the remote node and program requesting the link. The Connect Data Block also indicates how the remote program addressed this one (by name or object type code). If the program requires access control information from the remote program, such information would also be included as part of the Connect Data Block. The Connect Data Block and up to 16 bytes of user data are delivered to the user buffer specified in the Receive call.

3.8 Send/Receive Connect Confirm Message

The Send Connect Confirm Message call is used to accept a logical link requested by a remote program. That is, it is a positive response to a received Connect Initiate Message. The following information is specified with the call:

The user link address (ULA) by which the program will hereafter refer to this link.

The local link address (LLA) that identifies the link being accepted. (The LLA is taken from the Connect Initiate Message received for this particular link.)

The type of receive flow control desired by the calling program for incoming data messages over this link. (Flow control options are discussed in detail in Chapter 4.)

A receive maximum. The maximum amount of user data that the program wishes to receive as a unit (that is, as one Network Data Message).

Up to 16 bytes of user data.

A Receive call that returns a Connect Confirm Message from the pending message queue returns control information identifying it as such, along with requirements for messages to be transmitted over the link. The information includes the following:

The user link address (ULA) established by the receiving program in the Connect Initiate Message that requested the link. This identifies the particular link being confirmed.

The type of receive flow control requested by the remote program — that is, the type of flow control imposed on outgoing data messages. (Flow control options are discussed in detail in Chapter 4.)

A receive maximum. The maximum amount of user data that the local program will receive on this link in one Network Data Message. This will be the same or less than the size requested in the program's Connect Initiate Message. A smaller size is indicated by the local NSP if it cannot handle the size requested in the Connect Initiate Message.

A transmit maximum. The maximum amount of data the local program can send to the remote program as a unit (with one Network Data Message) over this link.

Up to 16 bytes of user data, delivered to the user buffer specified in the Receive call.

3.9 Send/Receive Connect Reject Message

A Send Connect Reject Message call is used to reject a logical link requested by a remote program. That is, it is a negative response to a received Connect Initiate Message. The following information is specified with the call:

The local link address (LLA) assigned by the local NSP to identify the link being rejected. (The local link address is taken from the received Connect Initiate Message.)

A code identifying the reason for the rejection. Ordinarily, this code is zero unless the program is rejecting the connection due to some standard reason, such as "invalid accounting data."

Up to 16 bytes of user data.

A Receive call that returns a Connect Reject Message from the queue returns control information identifying it as such and the following information:

A user link address identifying the link being rejected. This user link address will be the same as that established by the local program in the Connect Initiate Message that requested the link. The link could have been rejected by either the remote NSP or the remote program.

A code identifying the reason for the rejection if the link was rejected by the remote NSP or by the remote program for some standard reason, such as "invalid accounting data." (A list of NSP rejection codes is found in Appendix B.)

Up to 16 bytes of user data, if the link was rejected by the remote program for some nonstandard reason. This data will be delivered to the user buffer specified in the Receive call.

3.10 Send/Receive Network Data Message

The Send Network Data Message call transmits user information to a remote program over an established logical link. The logical link is identified by the user link address (ULA) established by the calling program in its Connect Initiate or Connect Confirm Message. The amount of user data that can be sent is limited to the transmit maximum established by the remote program (see the previous discussion of a received Connect Initiate or Connect Confirm Message). Sending a Network Data Message is subject to flow control imposed by the system and by the remote program, as indicated by the flow control option selected by the remote program. A program can request that link status information be returned by a Send Network Data call and use the information returned to schedule subsequent send and receive requests. A full discussion of flow control is deferred to Chapter 4.

A Receive call that returns a Network Data Message returns identifying control information and delivers the user data to a buffer specified in the call. The amount of user data received in one Network Data Message will not exceed the receive maximum established by this program for the logical link (see the previous discussion of a sent Connect Initiate or Connect Confirm Message). Incoming Network Data Messages are subject to system flow control and to the flow control option, if any, requested by the local program in its Connect Initiate or Connect Confirm Message. If the program requests flow control, it must send Link Service Messages to request data from the remote program (see Chapter 4).

3.11 Send/Receive Interrupt Message

The Send Interrupt Message call is used to transmit high-priority data to a remote program over an established logical link. All DECnet systems maintain separate data streams for Interrupt and Data Messages. The Interrupt Message will be delivered to the remote program ahead of all Data Messages waiting to be processed at the remote end. Up to 16 bytes of user data can be sent in an Interrupt Message. Sending an Interrupt Message is subject to flow control imposed by the remote system or program. The local program can request that link status information be returned following a Send Interrupt Message call and use the information returned to schedule subsequent send and receive requests. A full discussion of flow control is given in Chapter 4.

A Receive call that returns an Interrupt Message returns identifying control information and delivers up to 16 bytes of user data to a buffer specified in the call. The Interrupt Message will have been queued at the head of the pending message queue, behind the first Data Message and behind any other pending Interrupt Messages queued for the program. Incoming Interrupt Messages are subject to flow control imposed by the local NSP. After receiving each Interrupt Message, the local program must send a Link Service Message to reenable incoming Interrupt Messages from the remote program over a logical link (see Chapter 4).

3.12 Send/Receive Link Service Message

The Send Link Service Message call is used to request data over a logical link — either to request data from a remote program over a flow controlled link, or to reenable incoming Interrupt Messages from the remote program. In these two cases, detailed link status information can be returned or the call can be issued solely to obtain this information.

A Link Service Message is returned on a Receive call when a link becomes unblocked. If a local program has tried to send a Network Data or Interrupt Message and failed because the remote system or program has inhibited flow, the local NSP informs the local program when the condition clears. To encourage efficient processing of pending messages, NSP returns a Link Service Message on a Receive call only when the pending message queue is empty. Detailed link status information is returned that the local program can use in

regulating its send and receive requests. (The format of the status information for a received Link Service Message is the same as that returned, upon request, for a Send Network Data, Interrupt, or Link Service call.)

For a more complete understanding of the uses of Link Service Messages, see the discussion of flow control in Chapter 4.

3.13 Send/Receive Disconnect Message

The Send Disconnect Message call is used to break an established logical link provided that all Network Data, Interrupt, and Link Service Messages previously sent over the link have been acknowledged by the remote NSP. If they have not been acknowledged, the call will terminate with an error and the link will not be broken. Successful completion of the call means that no more messages can be sent over the link. The user link address is freed and can be used to establish another logical link. However, messages already in the pending message queue will remain. They can (and should) be cleared from the queue by doing selective Receive calls using the user link address of the disconnected link. Once the link has been disconnected, new messages received for the link will be discarded by NSP. The Disconnect Message is sent to the remote system. When it is received by the remote NSP, the link is broken at that end. Up to 16 bytes of user data can be sent with a Disconnect Message.

A Receive call that returns a Disconnect Message from the pending message queue returns control information identifying it as such, as well as a user link address and up to 16 bytes of user data. Receipt of a Disconnect Message indicates that the remote program has terminated the logical link. Furthermore, it indicates that all previous Network Data Messages, Interrupt Messages, and Link Service Messages sent by the remote program were placed in the pending message queue by the local NSP before the Disconnect Message was received. Once the Disconnect Message is received by the local NSP, the local program cannot send any more messages over the link.

The Disconnect Message is useful in "master-slave" communication when the master program only transmits data and the slave program only receives data. Since there is no two-way communication, the master program can issue a Disconnect Message to break the link with the assurance that the remote system has received all the data sent and queued it for the slave program. Successfully completing the disconnect does not guarantee that the slave has processed the data, however. Where such guarantees are desirable, or where two-way communication is desired, programs should agree to terminate the logical link by exchanging Network Data Messages, Interrupt Messages, or whatever.

3.14 Send/Receive Link Abort Message

A Send Link Abort Message is used to break a logical link immediately. Unlike a disconnect, no attempt is made to ensure that messages previously sent over the link have been acknowledged by the remote system. In fact, NSP discards any messages it has queued for transmission for the link. (Section

4.2.1 describes NSP transmit queuing.) No further data can be sent over this logical link. Received messages already in the pending message queue will remain but new messages received after the Link Abort Message is issued will not be queued for the link. Messages already in the queue can (and should) be cleared by doing selective Receive calls using the user link address of the aborted link.

If the link is established, the Link Abort Message is sent to the remote system as notification that the link is broken and up to 16 bytes of user data can accompany the Link Abort Message. A Link Abort Message can also be sent for a logical link that has not yet been confirmed by the remote program. In this case, NSP breaks the link at the local node, but it does not notify the remote system. (Hence, the user data does not reach the remote program.) Should the remote program confirm the link, the local NSP no longer knows the link and it so informs the remote NSP, which notifies the remote program. Should the remote program reject the link, the local NSP informs the remote NSP that it has no such link but since the remote program rejected the link anyway, no notification is given to the remote program.

A Receive call that returns a Link Abort Message from the pending message queue returns control information identifying it as such, as well as a user link address and a local link address to identify the link aborted. The local link address identifies a link aborted during the initial connection sequence. For example, if the local NSP received a Connect Initiate Message, but the local program has not yet responded with a Connect Confirm or Connect Reject Message, a Link Abort Message is queued with the same local link address as was specified in the received Connect Initiate Message. The link could have been aborted by the remote NSP or the remote program. If the link was aborted by the remote NSP, the message will contain a reason code identifying the reason for the rejection. If the link was aborted by the remote program, up to 16 bytes of user data can also be delivered with the Link Abort Message. Once the local NSP receives the Link Abort Message, the local program cannot send any more messages over the link.

Chapter 4
Network Addressing and Flow Control

With the background given in Chapters 2 and 3, we can now discuss two areas of interprogram communication in a DECnet network: network addressing and flow control. Network addressing concerns the methods whereby a program declares its identity for the purpose of receiving network messages. Flow control involves the various techniques used by DECnet to ensure an orderly transfer of message and control data between nodes in a network.

4.1 Network Addressing

When a program declares its intent to use the network message services, it is influenced by the following DECnet/E features:

- The options available in the Declare Receiver call
- The way in which DECnet/E NSP handles incoming requests for logical link connections
- The multiple copy execution feature, by which more than one copy of a network program can be executed in the RSTS/E timesharing environment
- The automatic job startup feature, by which DECnet/E allows programs to be defined by the system manager for automatic startup on receiving a request for a logical link connection

4.1.1 Declaring Network Names and Object Type Codes

Every DECnet/E program that wishes to receive messages from local senders or establish logical links for data exchange with other network programs must declare its identity in a Declare Receiver call. The program can declare its identity with a nonzero object type and with a null name — abbreviated to DR(n,null) in this chapter — with a zero object type and a nonnull name — DR(0,name) — or with a nonzero object type code and a nonnull name — DR(n,name). Remember from the discussion in Chapter 3 that nonzero object type codes should be used consistently throughout the network.

Declaring identity by object type code alone — DR(n,null) — permits multiple copies of a program to be executed without the necessity of generating a unique identity for each copy. DECnet/E allows more than one program to run and declare identity with the same object type code. Thus, several copies of a program can be executed simultaneously in response to incoming connect requests.

On the other hand, declaring identity by name alone — DR(0,name) — requires uniqueness in the sense that no other program at the local node can declare identity with the same name at the same time. If a program declares its identity with the same name each time it runs, multiple copies cannot execute simultaneously.

It is possible, however, to combine the features of uniqueness and of multiple copy execution. By appending the RSTS/E job number to a common root, a program can create a unique, nonnull name — for example, FAL21, FAL05. (The RSTS/E job number can be determined by a MACRO-11 program with the UU.SYS directive, described in the *RSTS/E System Directives Manual.*) The program can then declare its identity with both the name created *and* a nonzero object type code. This permits multiple copies of the program to be run as a result of incoming connect requests but permits each copy to create its own unique identity.

4.1.2 Handling of Incoming Connect Requests

When a Connect Initiate Message is received at the local node, NSP searches its list of Receiver ID Blocks for the proper receiver. (All other incoming messages reference an established logical link and will be queued to the correct program without searching the list of Receiver ID Blocks.) The method of addressing specified in the Connect Initiate Message determines how NSP processes the request.

Incoming requests for logical link connections can be addressed to either an object type code alone — CI(n,null) — or to a name alone — CI(0,name). DECnet/E does not allow incoming connect requests to be addressed using both object type code and name. It will reject such requests, sending back a Connect Reject Message with a reason code indicating that the requested program cannot be found.

Incoming Connect Request Addressed to Object Type Code — CI(n,null)
If the incoming Connect Initiate Message is addressed to an object type code — CI(n,null) — NSP checks its Receiver ID Blocks to see if one or more programs are running that have declared identity with this object type code. The Declare Receiver call could have specified either DR(n,null) or DR(n,name).

If one or more such receivers are found that have not reached their declared logical link maximum or message maximum, NSP queues the connect request for the first such receiver in its list (ordered by job number).

NOTE

Remember from Chapter 3 that a program sets its own limits on the number of logical links allowed and the size of the pending message queue in the Declare Receiver call.

If there is no such receiver or if there are one or more such receivers but all have reached their declared logical link maximum or message maximum, NSP attempts to start another copy of the program. This process is described fully in Section 4.1.4 and also in the *DECnet/E System Manager's Guide*. Briefly, NSP checks to see if the program identified in the Connect Initiate Message has been defined by the system manager for automatic startup. If so, NSP creates a job and starts the program. The program must then issue a Declare Receiver call with the appropriate object type code. This creates a Receiver ID Block and a pending message queue to which the waiting Connect Initiate Message can then be transferred. If the program does not issue a Declare Receiver call within two minutes, NSP rejects the link by returning a Connect Reject Message to the remote program.

Incoming Connect Request Addressed to Name — CI(0,name)

When the incoming Connect Initiate Message is addressed to a name — CI(0,name) — NSP checks its list of Receiver ID Blocks to see if a program is running that declared its identity with this name. The Declare Receiver call could have specified either DR(0,name) or DR(n,name).

If such a receiver program is running, and it has not yet reached its declared logical link maximum or message maximum, the Connect Initiate Message is placed in the receiver's queue of pending messages.

If no such program is running, NSP will attempt to start the program. If it is successful in doing so, the program started must declare its identity with the appropriate name. Again, if the program fails to issue a Declare Receiver call within two minutes, NSP rejects the connection.

If such a program is running, but it has reached its declared logical link maximum or message maximum, NSP rejects the connection, returning a Connect Reject Message with an appropriate reason code explaining the rejection. Automatic startup is not attempted in this case, because a program is already running with the name to which the Connect Initiate Message was addressed. A second copy with the same name cannot be generated since receiver names must be unique.

4.1.3 Multiple Copies

In a DECnet/E system, multiple copies of a program can be started by a local terminal user, through the RSTS/E BATCH processor, or through the automatic job startup feature. The manner in which copies of a program can be started has some bearing on the way in which a Declare Receiver call should be issued.

Multiple Copies Initiated by Terminal Users

A program run from a local terminal with the purpose of communicating with a remote program will normally send at least the initial Connect Initiate Message requesting a logical link. (It is unlikely that remote users would wish to have the success of incoming link connections rely upon someone starting the program manually.)

In this case, the program can declare its identity with DR(n,null), or with DR(n,name) or DR(0,name) as long as the program generates a unique name with each copy, as described in Section 4.1.1. The choice depends on whether or not the program receives any connect requests, and if so, if it is necessary for the remote program to connect with a specific copy.

Suppose, for example, that the local program issues the initial Connect Initiate Message for establishing the first logical link. Suppose further that it is then necessary for the remote program to establish a second logical link with the same copy that sent the Connect Initiate Message for the first link. The local program can declare its identity with either the form DR(0,name) or DR(n,name). The unique copy name is passed to the remote program in the Connect Data Block in the Connect Initiate Message. The remote program can then address its connect request in the form CI(0,name) to ensure that the request is delivered to the same copy that issued the first Connect Initiate Message.

Multiple Copies Executed under BATCH

Multiple copies of a program can also be executed through use of the RSTS/E BATCH processor (see the *RSTS/E System User's Guide*). Again, such programs will normally be responsible for initiating at least the first logical link connection. The techniques described previously for multiple copies started by terminal users would also apply here.

Multiple Copies Initiated through Automatic Startup

Multiple copies of a program can also be initiated for programs that the system manager has designated for automatic job startup. Such copies result from incoming connection requests addressed CI(n,null). (See Section 4.1.4 for full details.) The local program can declare its identity with DR(n,null), or with DR(n,name) as long as the name is unique for each copy.

4.1.4 Automatic Job Startup

Using the Network Control Program (NCP), the system manager can install user programs for automatic startup. Such programs reside as files on disk in the normal fashion and can be started by NSP to fulfill incoming requests for logical link connections (see Section 4.1.2). This installation method is described fully in the *DECnet/E System Manager's Guide*. Briefly, a DEFINE OBJECT or SET OBJECT command in NCP relates the filename to, again, an object type code, a name, or both. These commands also allow two parameters to be associated with a program. The second parameter is made available to the program when it is started automatically. (The first parameter is used as a starting line number for BASIC-PLUS or BASIC-PLUS-2 programs. It is ignored for MACRO programs.)

NSP maintains a list of these definitions in its parameter file. The definitions are ordered by object type code, and thus, there can be only one program defined for automatic startup for each object type code. In particular, there can be only one program defined for automatic startup with an object type code of zero — (0,name).

When a program must be started automatically, NSP checks the object type code or name given in the incoming connect request against its list of defined programs. If a match is found, NSP executes a monitor request to run the program as a detached job under RSTS/E.

Once started, the program must declare its identity with the object type code or name that was specified in the Connect Initiate Message that caused the program to be started. If it does not, the connect request will never be delivered to the proper pending message queue.

NSP passes this information to the started program in core common. Any optional parameter established by the DEFINE OBJECT or SET OBJECT command is also passed in the string. A MACRO program can access core common directly, as described in the *RSTS/E System Directives Manual*. The format of the core common string is as follows:

Byte(s)	Contents
CORCMN+0	Length of data in core common
+1	Object type code (0-255)
+2-7	Name (ASCII)
+8-9	Second DEFINE parameter (-32768 through 32767)
+10	Reserved

The mnemonic CORCMN is assigned by the file COMMON.MAC to octal location 460 (see the *RSTS/E System Directives Manual*).

If the object type code in byte 1 of core common is nonzero, the program must declare its identity with that object type code. It can also use a nonnull unique name, if desired. If the object type code is zero, the program must declare its identity with the name given in bytes 2-7 of core common. It can also declare its identity with a nonzero object type, if desired.

When the Declare Receiver call is issued, NSP transfers the waiting connect request to the program's queue of pending messages as a Connect Initiate Message. NSP waits two minutes from the time it starts the program for the connect request to be transferred.

If the connect request is not transferred to a queue within this time period, NSP rejects the connection, sending a Connect Reject Message to the remote program. Since the remote program could also abort the link within this period, the first action of an automatically started program (after issuing the Declare Receiver call) should be to issue a Receive call to check for pending messages. If there are none, the program should issue a Remove Receiver call, and terminate execution with the Kill Job system call. (See the UU.CHU system call in the *RSTS/E System Directives Manual*.) Once the connect request has been received, the program should respond to the request immedi-

ately. While the local NSP's timer is stopped when the message is transferred to the receiver's pending message queue, the remote node could be executing a timeout on the Connect Initiate Message.

A program that has been started in response to an incoming Connect Initiate Message runs as a privileged RSTS/E user, and it is the responsibility of that program to protect the rest of the system from unauthorized outside access. Thus, after receiving the Connect Initiate Message, the program should check the accounting information passed in the Connect Data Block to determine the user account (and, therefore, the privilege level) in which it should run. If no such information is specified in the Connect Data Block, the program should issue a Get Local Node Parameters call to obtain the local node's default project-programmer number, if any. If a default account has been specified, the program should then look up the password for that account using the Read Accounting Data system call and log into the default account. (See the UU.RAD and UU.LIN system calls in the *RSTS/E System Directives Manual*.) If the local node has no default account, the connect request should be rejected. (Programs that always log into a predetermined account need not go through this procedure.)

4.1.5 Examples of Network Addressing

This section presents three examples of network addressing using the features just discussed.

The first example (Figure 4-1) shows the network addressing used by two DECnet/E utility routines — the Network File Transfer utility (NFT) and the File Access Listener (FAL). NFT is run by local users to connect with a remote FAL to access files at remote nodes. FAL is run at the local node as a result of incoming connect requests from a remote NFT. Both programs are designed to execute multiple copies as demand requires.

Since NFT is started only by local terminal users and never by remote request, an object type code is not necessary. NFT is loaded from the system disk and declares its identity with an object type code of zero and a unique name each time it is run. NFT issues the Connect Initiate Message to establish a logical link for the transfer of data to or from remote nodes as specified by the user.

Copies of FAL are started by incoming Connect Initiate Messages addressed to object type code 17. FAL declares its identity with object type code 17 and a unique name. It also declares that the maximum number of logical links allowed is one so each incoming connect request starts a new copy.

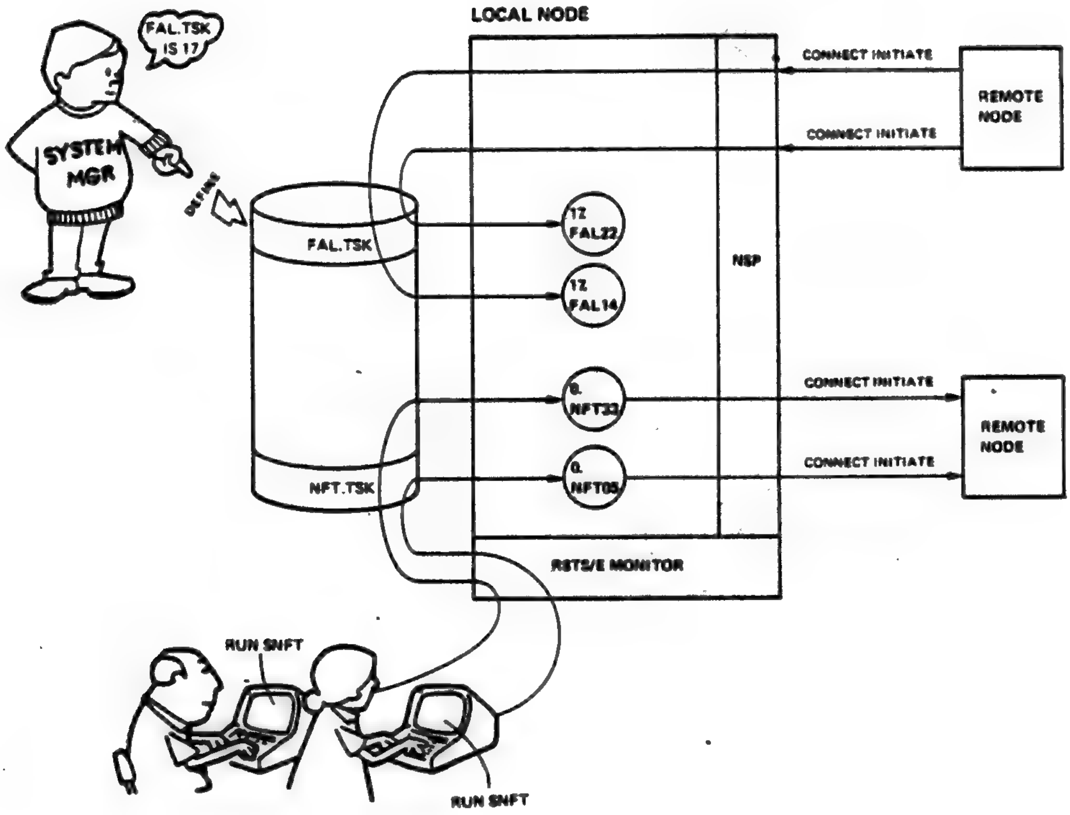


Figure 4-1: Network Addressing Used by NFT and FAL

The second example (Figure 4-2) shows NETCPY, a device copying program that can be started either by local terminal users or by incoming connect requests. The program copies an entire device between nodes. Multiple copies are executed according to demand. Here, the Declare Receiver call is different depending on how the program is started. NETCPY is a BASIC-PLUS program and takes advantage of the starting line number feature. Using the Network Command Program (NCP), the system manager issues a command to SET or DEFINE NETCPY.BAC as object 20, with a starting line number of 29000. If NETCPY is started by a Connect Initiate Message from a remote program addressed to object type code 20, the program declares its identity with object type code 20, a unique name, and a link maximum of one. If it is started by a local terminal user, it declares its identity with an object type code of 0, a unique name, and a link maximum of zero. Declaring a zero link maximum ensures that a program started by a local user will never have a Connect Initiate Message queued from a remote program.

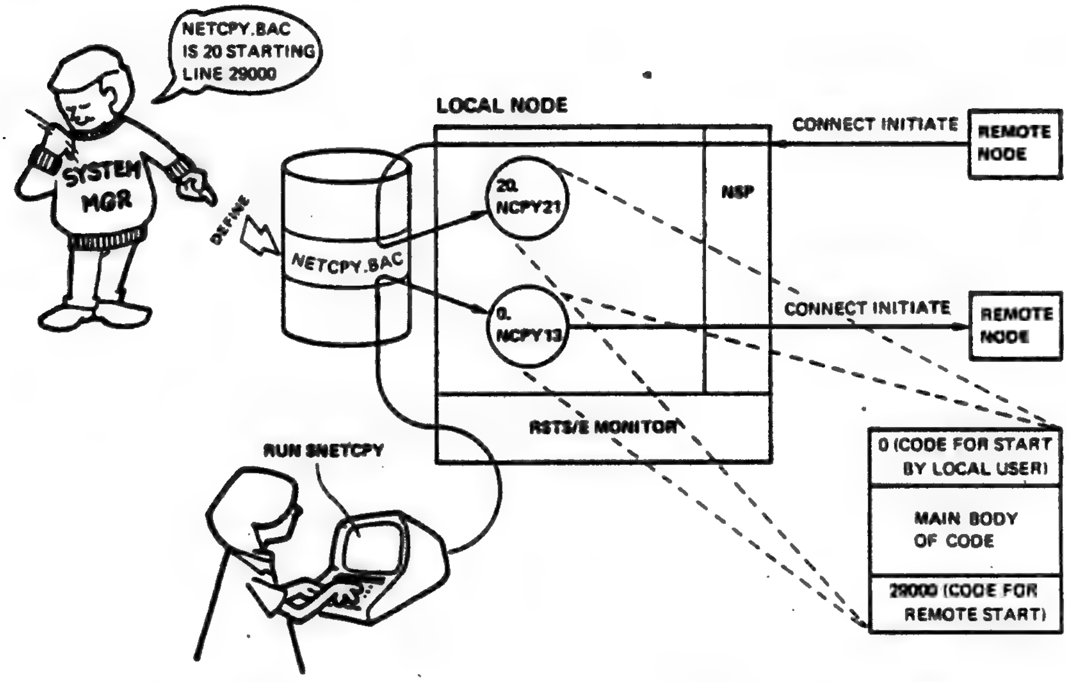


Figure 4–2: Network Addressing Example — Different Declare Receiver for Local and Remote Start

The third example (Figure 4–3) shows a program that can be run either by a local terminal user or as a result of an incoming connect request. The local terminal user is presented with a data display. Parameters are selected by the user and passed to a remote program monitoring a data gathering device. At the end of the day, the remote program passes the data back to the local program for analysis and to the local terminal user, who repeats the process.

Since only one copy of the program is ever needed at any given time, it declares its identity with a zero object type code and the name STAT. It is stored as file RSTAT.SAV on disk, where it can be run by the local user. Using NCP, the system manager has issued a command to SET or DEFINE RSTAT.SAV as STAT.

4.2 Flow Control

DECnet flow control protects user programs as well as entire systems from being flooded with more messages than can be processed at any given time.

A DECnet/E sender is subject to flow control regulations imposed by the local NSP, the remote NSP, and the remote program (if the remote program requested flow control when the link was established). As a receiver, a DECnet/E program can select from three flow control options for a logical link to regulate the amount of data sent to it by the remote program.

As a sender and/or receiver, a program must deal with the following four aspects of DECnet/E operation:

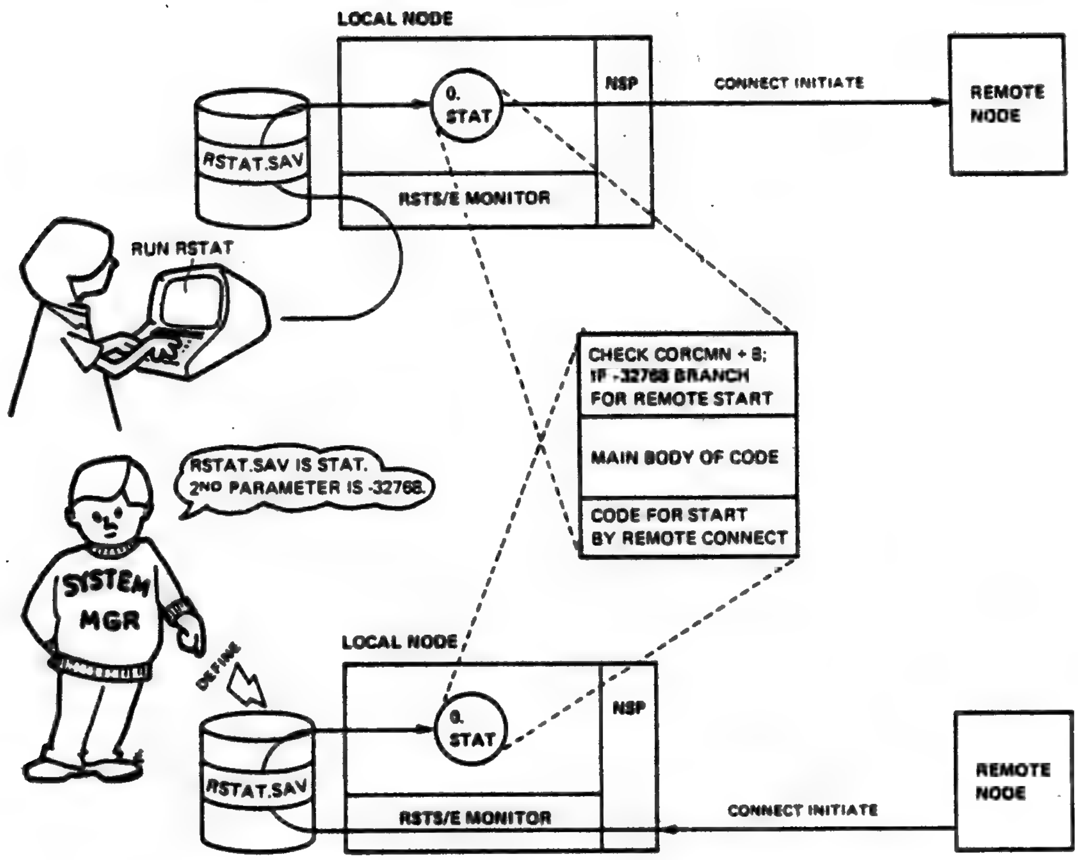


Figure 4-3: Network Addressing Example — Declaring Identity by Name Alone

- **Transmit queue management**, imposed by the local (sending) NSP, ensures that a local program does not use too much system buffer space for transmission.
- **Backpressure flow control**, imposed by the remote NSP, is used as a safeguard against excessive incoming data messages.
- **Data Message flow control**, imposed by receiving user programs, enables a program to control the flow of data to it. Both programs using a logical link can (independently) choose to receive network messages only when they specifically request data from the other program; or each can choose no flow control, placing no restrictions on the amount of data that they will receive.
- **Interrupt Message flow control**, imposed by either the remote NSP or the remote program (depending on the DECnet implementation), protects against Interrupt Message congestion. A DECnet/E program can receive only one unsolicited Interrupt Message over a link. After that, the program must request an Interrupt Message before the remote program can send one.

4.2.1 Transmit Queue Management

Once a logical link is established, DECnet/E maintains two transmit queues for the link: a transmit queue for Network Data Messages and a transmit

queue for Interrupt and Link Service Messages. Both queues hold messages waiting for acknowledgment from the remote NSP.

For example, when a DECnet/E program sends a Network Data Message to a remote program over a logical link, the message is queued to the Transport software for transmission and is also placed in the data transmit queue. The Network Data Message is held there until the local NSP receives an acknowledgment from the remote NSP indicating that it has received the message.

If NSP does not receive an acknowledgment within a reasonable period of time, it assumes that the message has been lost and requeues it to Transport for retransmission. If the message has not been successfully transmitted after a specific number of retries, NSP aborts the link with a reason code indicating that the remote system is not responding.

The timeout period used depends on the message type. For outgoing Connect Initiate Messages, a predetermined value is used. Once a logical link is established however, network management dynamically computes the timeout period, using various network parameters defined by the system manager using NCP and the SET EXECUTOR command. (See the *DECnet/E System Manager's Guide* for details.)

This acknowledgment procedure between the two NSPs ensures that messages sent over the logical link have been received, and that they were received in the same order they were sent.

Each of the transmit queues for a logical link can hold up to eight messages. (This limit can be changed by the system manager to any number from one to eight to further protect system buffer space.) When one of the transmit queues for a logical link is filled with messages waiting for acknowledgment, further transmission of that type of message over the logical link is inhibited by the local NSP until one or more of the queued messages have been acknowledged by the remote NSP.

An error is returned to the program if it issues a send while the transmit queue for that message type is full. This condition is temporary. When the condition clears, sends are again allowed for that message type. A program getting such an error on a send should simply wait a short time (1 to 5 seconds) and retry the send.

The state of the transmit queues also affects the sending of Disconnect Messages. If any messages are waiting for acknowledgment in the Data or Interrupt/Link Service transmit queue, a Send Disconnect Message call for that logical link will not be allowed and will result in an error. Thus, a program sending a successful Disconnect Message can assume that all previously sent messages have been received and acknowledged by the remote NSP. A Link Abort Message for that logical link will be allowed, however, and any messages waiting in the transmit queues will be discarded.

Figure 4–4 illustrates transmit queue flow control. PROGA has established two logical links, and PROGB and PROGC have each established one logical link. Each logical link has its own Data and Interrupt/Link Service Message transmit queues. PROGA issues a Send Network Data Message system call.

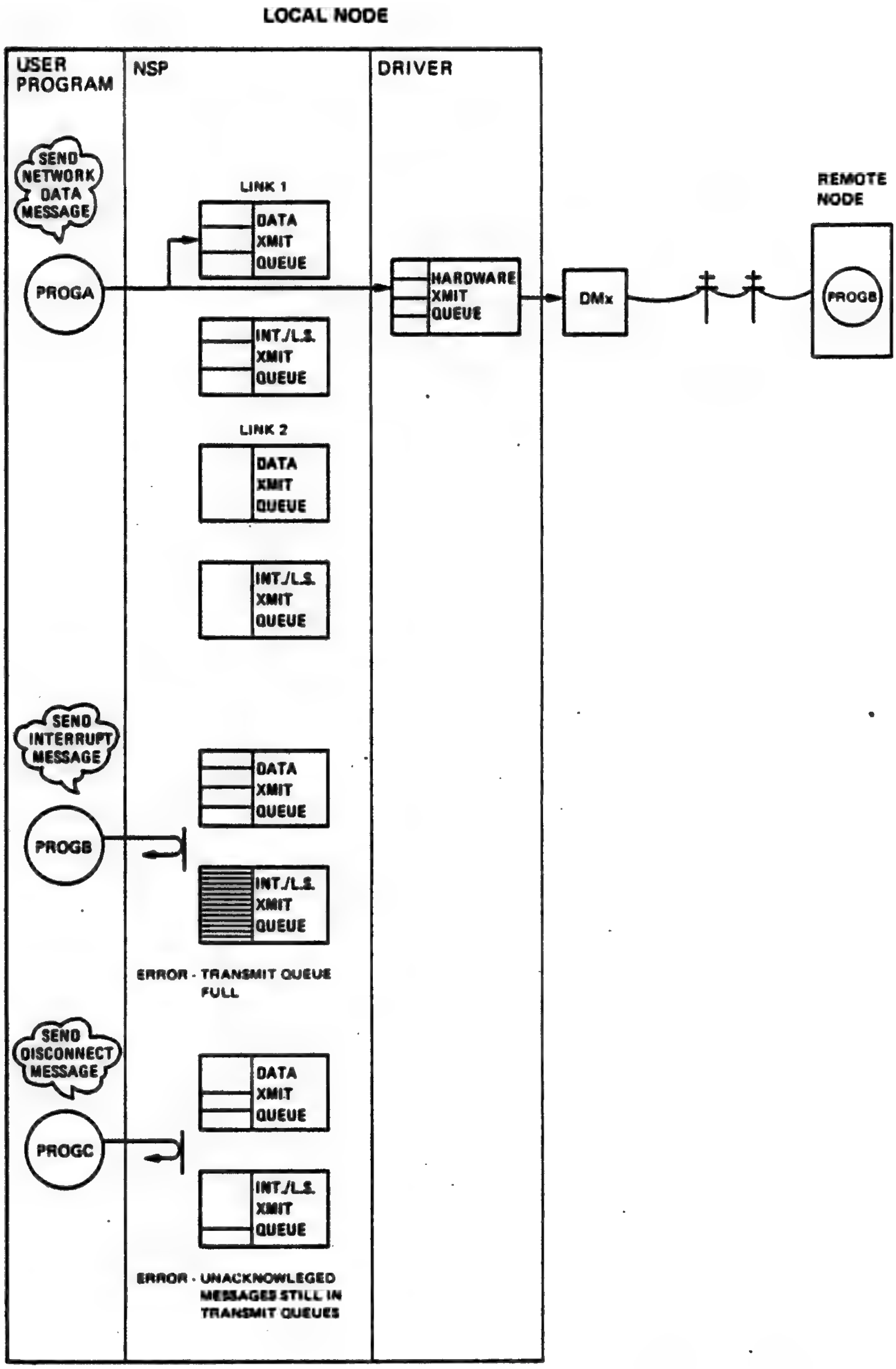


Figure 4–4: How Transmit Queue Flow Control Affects a DECnet/E Program

The message goes to the Data Transmit Queue for that logical link and is also queued for transmission by the communication hardware. At the device driver level, all network messages (both data and control messages) await transmission over the physical line.

PROGB is shown trying to send a Network Interrupt Message. The Interrupt/Link Service transmit queue for that link is full, however, so NSP returns an error on the send. PROGC issues a Disconnect Message for its logical link. Since messages are still waiting for acknowledgment in the transmit queues, the Disconnect Message is not allowed and an error is returned. (Sending a Link Abort Message would have broken the link but the messages would have been discarded.)

4.2.2 Backpressure Flow Control

NSP at the remote node can inhibit the local program from sending Network Data Messages over a logical link. The reasons for this blocking can vary from system to system: lack of system buffer space, system load, or some similar condition. This type of flow control, whereby the *remote* NSP can inhibit message flow from the local program, is known as backpressure.

When the remote NSP receives a Network Data Message that it cannot process, it returns a negative acknowledgment (NAK) for the message, along with an error code informing the local NSP that data message flow to the remote program is being inhibited on the link over which the message was sent. The local NSP then inhibits transmission of Network Data Messages over the indicated logical link until further notice is received from the remote NSP. (Note that the local program can still receive messages over the logical link.) When the remote NSP informs the local NSP that the backpressure condition has cleared, the local NSP retransmits the message that was previously NAKed and reenables transmission of Network Data Messages over the logical link.

If the backpressure condition clears before the local program attempts another send over the link, the program is not informed that the conditon occurred. However, if the program attempts to send another Network Data Message while transmission on the link is still inhibited, the local NSP returns an error to the program (see A, in Figure 4-5). When the condition clears, the local NSP delivers a Link Service Message to the program, informing it that transmission of Network Data Messages is again allowed for the indicated logical link (see B, in Figure 4-5). This Link Service Message is delivered to the local program on a Receive call when the queue of pending messages is empty.

Note that the remote NSP enforces flow control for specific logical links, not for the entire node.

From the reverse viewpoint, the local NSP will inhibit incoming data messages on a logical link when a Network Data Message is received from a remote program over the link and the pending message queue of the intended receiver is full. (Remember that the local program itself limits the size of its

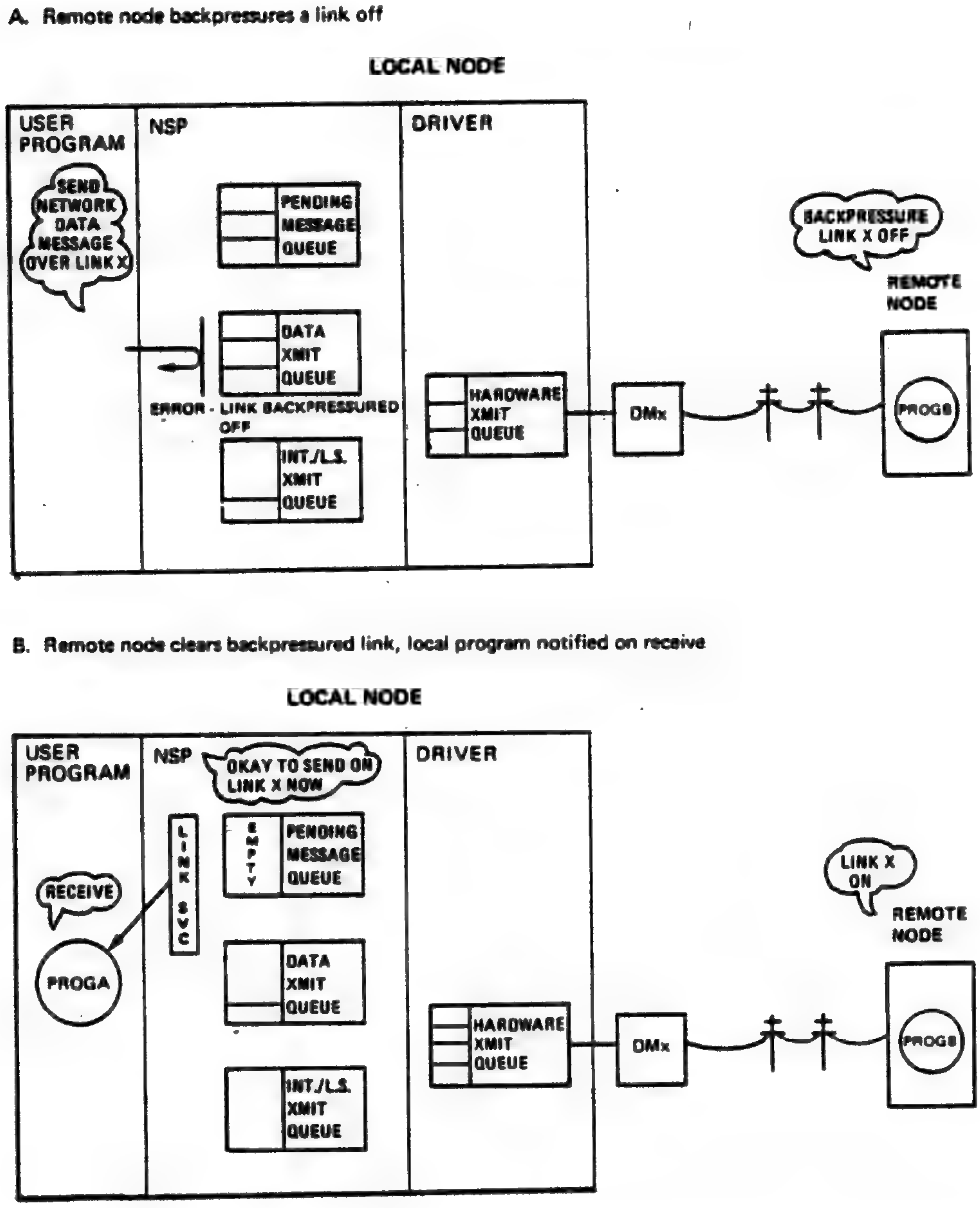


Figure 4-5: How Backpressure Flow Control Affects a DECnet/E Program as a Transmitter

pending message queue in its Declare Receiver call.) The local NSP informs the remote NSP that incoming data message flow has been inhibited by sending a message to the remote system.

Under these conditions, the Network Data Message is not queued for the local program, but is NAKed by the local NSP. This forces the remote NSP to retransmit the message after flow is reenabled. Any data messages already transmitted by the remote node will also be NAKed by the local NSP.

Furthermore, the local NSP marks all other logical links on the Receiver ID Block to be turned off if any Network Data Messages are received on those links. However, no message is sent to the remote system unless a Network Data Message is received on one of these links before the restriction is lifted. (Note that while the local NSP will no longer accept incoming messages, the local program can still transmit.)

When the local program empties its pending message queue, any links that were actually turned off are turned back on and a message is sent to the remote NSP to reenable transmission. Those links that were merely flagged to be turned off are restored to normal operation, but no message is sent to the remote system.

4.2.3 Program-Regulated Flow Control

During the connection sequence that establishes a logical link, each program must select one of the following three flow control options to regulate incoming data over the link:

- **Segment flow control**. The program requests data from the remote program by indicating how many segments it will accept at any given time. A segment is defined as the amount of user data transferred in one Network Data Message. The maximum size of a segment is determined by the receive maximum established in the Connect Initiate or Connect Confirm Message. The sending program is not required to use this maximum, however, and can, in fact, send smaller segments. Note also that segments transmitted and received by the same program are not necessarily equal in size.
- **Logical message flow control**. The program requests data from the remote program by indicating how many logical messages it will accept at any given time. A logical message can consist of one or more data segments. Control flags, included as part of a Network Data Message, indicate whether the data segment is the beginning, middle, end, or sole segment of a logical message.
- **No active flow control**. The program sets no limits on the rate at which messages can be sent to it over a logical link.

If the program selects segment flow control or logical message flow control, the remote NSP will not send Network Data Messages over the link until the local program requests such transmission. Requests are made by sending a Link Service Message requesting a specific number of segments or logical messages.

If the program selects the third option, no active flow control, no Link Service Messages can be sent except those requesting Interrupt Messages. Network Data Messages will be transmitted by the remote NSP as the remote program sends them, subject to backpressure.

Flow control selection is independent at both ends of the logical link. That is, the remote and local programs need not select the same type of flow control. The option selected by the remote program, however, affects how the local program can send Network Data Messages. Therefore, the following discussion of flow control is presented from two viewpoints:

1. How the option selected by the local program affects incoming Network Data Messages
2. How the option selected by the remote program affects the way the local program sends Network Data Messages

Flow Control on Incoming Data

If the local program selects logical message flow control in the Connect Initiate or Connect Confirm Message establishing a logical link, the local program must request data from the remote program by sending Link Service Messages specifying the number of logical messages to be transmitted.

When the Send Link Service Message call is issued, the local NSP adds the value specified to a message counter maintained for the link. The local NSP also sends the information on to the remote NSP. The remote NSP likewise adds the value specified to a message counter it maintains for the link. Thus both NSPs count the outstanding message requests. Each time the remote NSP sends the end segment of a logical message, it decrements the message counter. When the counter reaches zero, the remote NSP will inhibit transmission until the local program sends another Link Service Message specifying another value to add to the message counter.

The local NSP uses its message counter as a protective measure. Each time it receives a data segment with the end-of-logical-message flag set, it decrements the message counter. When the counter reaches zero, the local NSP no longer accepts data segments over the link. Should the remote NSP send a segment while the local NSP's counter is zero, the message is discarded and the link is aborted. The local NSP places a Link Abort Message in the local program's queue of pending messages and sends a Link Abort Message to the remote NSP. (This situation is a protocol violation and should not occur with correct NSP operation.)

When the local program selects segment flow control in the Connect Initiate or Connect Confirm Message, the local and remote NSP maintain segment counters for the logical link. The remote program can still send segments with the logical message control flags set for the local program to process internally, but the counters maintained by NSP are kept for data segments sent, not for logical messages. Data is requested by sending a Link Service Message indicating the maximum number of segments to be sent. The local and remote NSPs add this value to their data segment counters and then decrement the

counters as the remote program sends segments. When the data segment counter reaches zero, transmission over the logical link is suspended until another Link Service Message increments the count.

When no active flow control is requested in the Connect Initiate or Connect Confirm Message, segments from the remote program can be transmitted over the logical link without being requested by the local program. The local and remote NSPs do not keep counters to regulate data sent and Link Service calls are not used to request transmission of data. Data Message flow can still be restricted by backpressure, however.

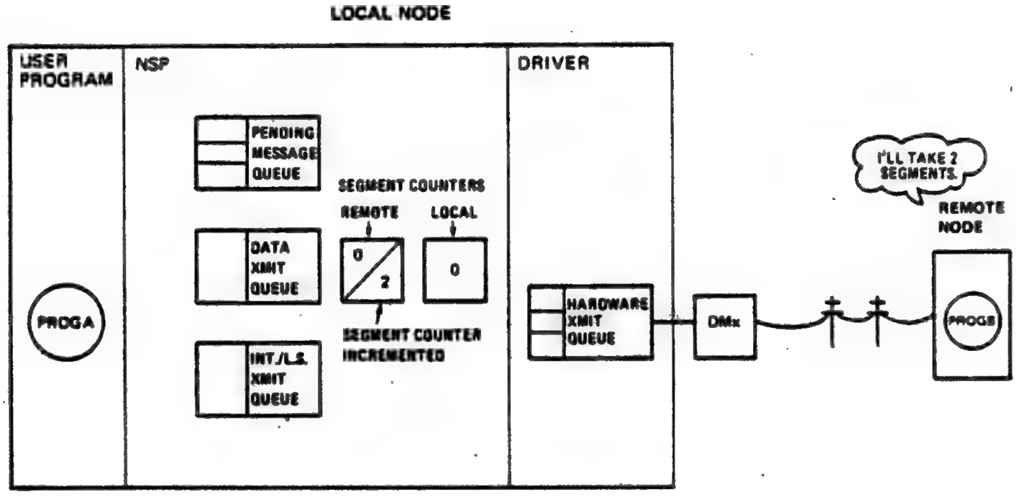
Flow Control on Outgoing Data

When the remote program specifies a flow control option in its Connect Initiate or Connect Confirm Message establishing the logical link, the local NSP notes the option and establishes the appropriate counter. The counter is increased when a request for data is received from the remote system or program. If the remote program requested segment flow control, the counter is decremented by one each time the local program issues a Send Network Data Message call. If the remote program requested logical message flow control, the message counter is decremented by one each time a Network Data Message is sent that is flagged as the end data segment of a logical message.

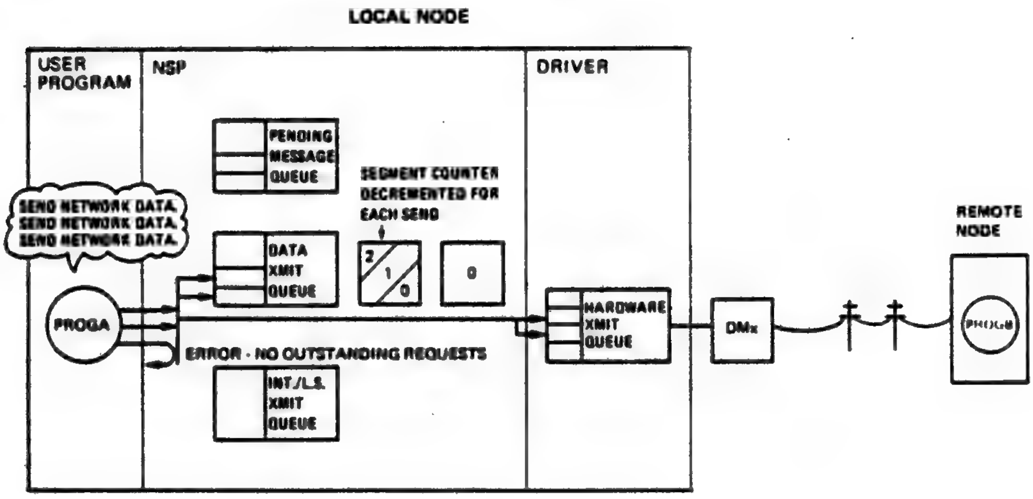
When the counter equals zero, the local NSP inhibits further transmission of Network Data Messages over the logical link, returning an error if the local program tries to do so. In this circumstance, the local NSP delivers a Link Service Message to the local program when the condition clears and the pending message queue is empty. A Link Service Message is not delivered, however, if the local program did not attempt to send a Network Data Message while the request count was zero.

This process is illustrated in Figure 4–6. The remote program has requested segment flow control. First, the remote program requests two segments (see A, in Figure 4–6). The segment counter for the remote program is incremented, but since PROGA has not had transmission inhibited, no Link Service Message is queued. The local program issues three Send Network Data Message calls (see B, in Figure 4–6). The segment counter is decremented to one with the first send, and to zero with the second send. The third send, while the counter is zero, results in an error. When the remote program receives and processes its two segments, it asks for two more segments (see C, in Figure 4–6). The segment counter is incremented from 0 to 2, reenabling data sends. Since PROGA attempted a send that failed, a Link Service Message is delivered to PROGA to indicate that data flow over the link has been reenabled. The Link Service Message is delivered to PROGA when it executes a Receive call when the pending message queue is empty.

A. Remote program requests two segments



B. Local program sends two segments, third send returns an error



C. Remote program requests two segments, local program notified on receive

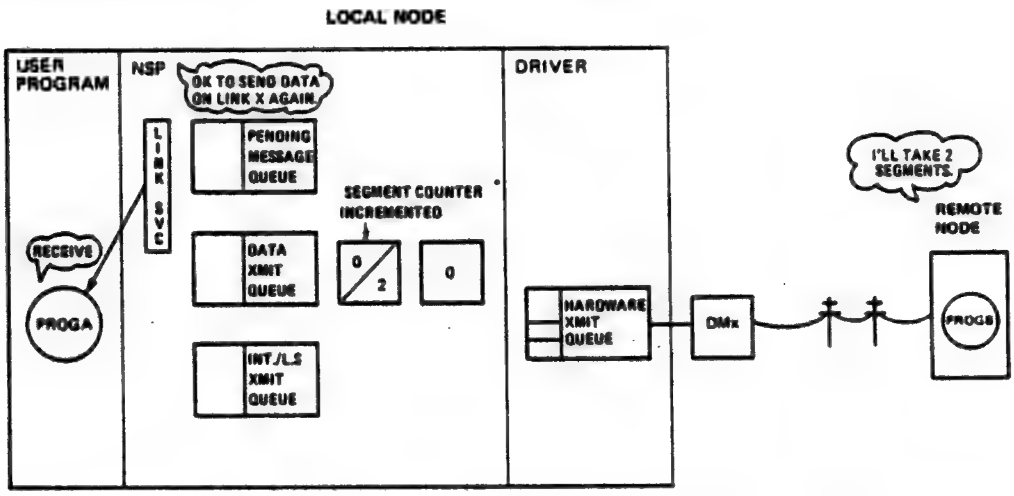


Figure 4-6: How the Flow Control Option Selected by the Remote Program Affects a DECnet/E Program as a Transmitter

4.2.4 Interrupt Message Flow Control

The transmission of Interrupt Messages over a logical link is subject to flow control by a request count mechanism. This mechanism is controlled by either the remote NSP or the remote program, depending on the particular DECnet implementation.

All logical links are initiated with an outstanding Interrupt Message request count of one from the remote system. If the local program sends an Interrupt Message over the logical link, this count is decremented to zero. The local NSP does not allow another Interrupt Message to be sent over this logical link until the remote NSP or remote program sends a Link Service Message, incrementing the request count. An attempt by the local program to send an Interrupt Message while the counter is zero will result in an error.

Once the request count is incremented by the remote system or program, the local NSP reenables Interrupt Message flow. If the local program has tried to send an Interrupt Message while the count was zero, NSP notifies the local program with a Link Service Message that Interrupt Messages can again be sent over this logical link. This Link Service Message is delivered to the local program on a Receive call when the pending message queue is empty. If the local program has not tried to send an Interrupt Message over the link while flow was disabled, no Link Service Message is delivered.

From the reverse viewpoint, all DECnet/E logical links are initiated with an outstanding request for one Interrupt Message. If the remote program sends an Interrupt Message over the link, the local NSP decrements this count to zero. No further Interrupt Messages are accepted until the local program requests one by sending a Link Service Message to increment the count. This also increments the request counter at the remote end and thus allows the remote program to send an Interrupt Message if it so desires. DECnet/E NSP does not allow the local program to request an Interrupt Message for a logical link unless a previous Interrupt Message on the link has been received.

In DECnet/E, only one Interrupt Message can be requested on a logical link at any one time. That is, the request count is either zero or one. If a program tries to issue a Link Service Message requesting an Interrupt Message on a link when the request count is already one, the Link Service Message will result in an error.

Figure 4-7 illustrates Interrupt Message flow control for a DECnet/E receiver. The remote program sends an Interrupt Message over a logical link (see A, in Figure 4-7). The local NSP decrements the Interrupt Message counter to zero and does not accept further Interrupt Messages over this link. The remote NSP also decrements its own counter and inhibits further Interrupt Messages from being sent. The local NSP maintains its counter as a safeguard measure. The local program sends a Link Service Message requesting one Interrupt Message over this logical link (see B, in Figure 4-7). The local NSP increments its counter and tells the remote NSP to reenable Interrupt Messages over the link. The local program must receive the first Interrupt Message before the Link Service Message can be sent.

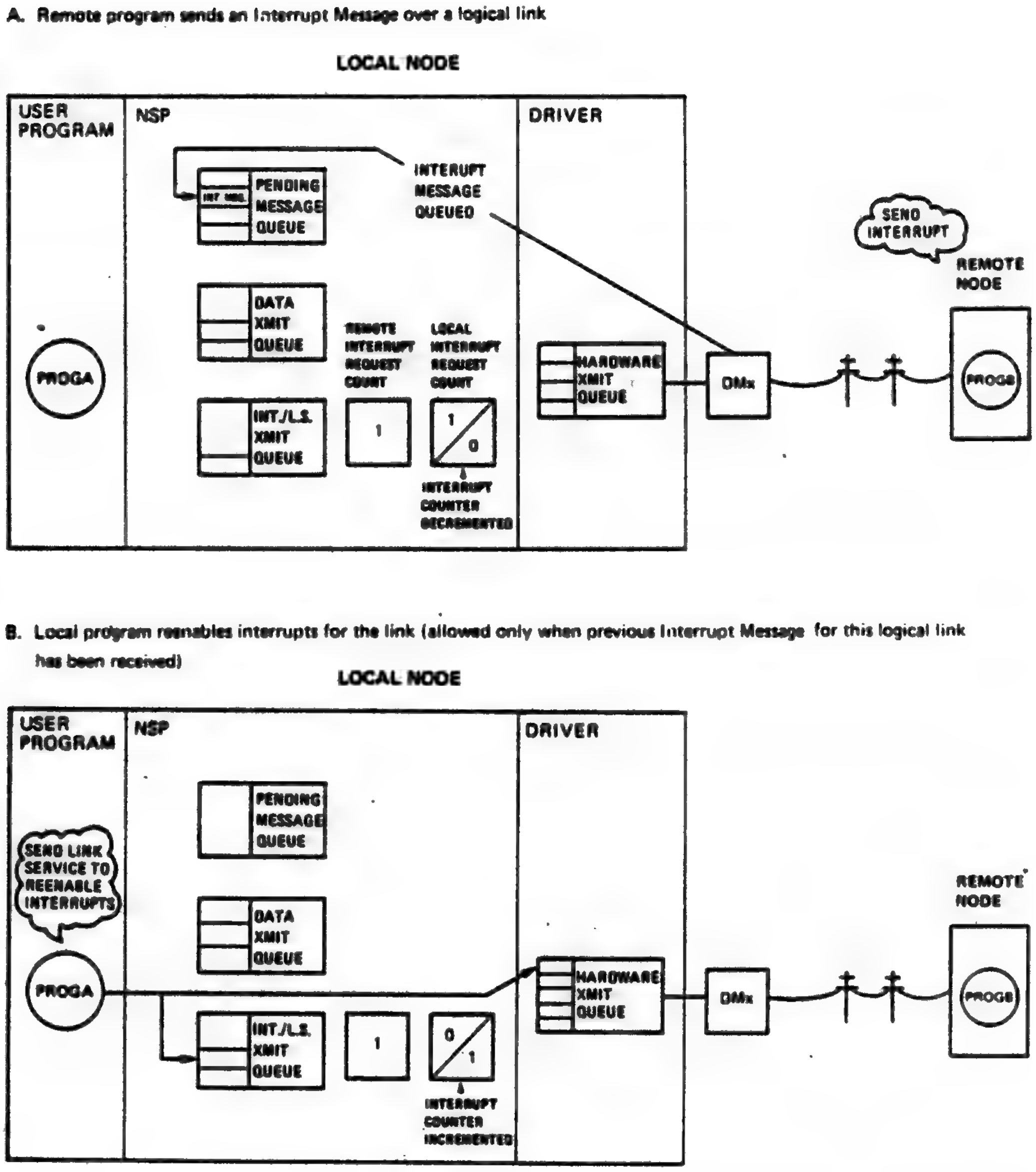


Figure 4–7: How Interrupt Message Flow Control Affects a DECnet/E Program as a Receiver

4.2.5 Programming Hints for Flow Control

A DECnet/E program can be prevented from transmitting for a number of reasons:

1. No buffer space is available at the local node. (Prohibits sending of all message types.)
2. The Data transmit queue is full. (Prohibits sending of Data Messages.)
3. The Interrupt/Link Service transmit queue is full. (Prohibits sending of Interrupt and Link Service Messages.)

4. Message flow over the link has been inhibited by the remote system because of a backpressure condition. (Prohibits sending of Data Messages.)

5. No requests are outstanding for segments or logical messages from the remote program. (Prohibits sending of Data Messages.)

6. No requests are outstanding for Interrupt Messages. (Prohibits sending of Interrupt Messages.)

The first three conditions indicate congestion at the local node. They are temporary and no notification is given to the program when the condition clears. When one of these errors occurs on a send call, the program should simply reissue the call after a short delay of one to five seconds.

The last three conditions, however, indicate congestion at the remote node. Since there is no way to predict how long it can take for one of these conditions to clear, the local NSP does not allow further send requests of Data or Interrupt Messages until the remote system indicates that the condition has cleared.

When sending is blocked for one of these last three conditions, any attempt to send a message by the local program results in an error. The program can still receive messages, however. If an attempt to send a message resulted in an error in this fashion, NSP notifies the program when the condition clears. It does this by delivering a Link Service Message to the program when the program issues a Receive call and the pending message queue is empty. No notification is given if the local program did not try to send a message.

Thus a program whose main function is to transmit data can be designed to send data over a logical link until transmission is blocked by one of the last three conditions. When sending is inhibited, the program can issue Receive calls to process all pending messages. When the pending message queue is empty and all conditions that block flow have cleared, a Link Service Message is delivered on a Receive call, giving current status information on flow control for the logical link. (If more than one logical link has been blocked, Receive calls should be issued for all links until no messages are returned, since link service information can also be available on the other links.)

A program whose main function is to receive data is concerned with keeping the pending message queue emptied so that the queue does not fill and cause the local NSP to inhibit incoming message flow over the program's logical links. The Receive call with sleep can be used effectively by such a program so that it will be notified when a message is delivered to the queue. In this case, a second Receive call must be issued to retrieve the message.

An interactive program can choose between these mostly-send and mostly-receive approaches, according to the requirements of the application. A program can also take advantage of the link status information that can be returned after sending Network Data, Interrupt, and Link Service Messages to regulate send and receive requests. The link status information returned is in the same format as a received Link Service Message.

Part III
Network Programming in MACRO

Chapter 5
Network Programming in MACRO

5.1 Programming Background

To use RSTS/E message services in a MACRO assembly language program, you code calls that are expanded at assembly time from a library file. The library file must be included with your source files for the assembly to produce an object module with the expansions. The expansions, in turn, call subroutines from another library file that must be included with your assembled object file at link time to produce an executable program. Thus, there are three aspects to using the message calls in a MACRO program: coding, assembling, and linking.

5.1.1 Coding

The message macro expansions use internal macros that must be defined in your program. The initialization macro .MSR (without any arguments) defines the necessary internal macros and must itself be established (with .MCALL) and invoked before any of the message calls are used. For example,

```
.MCALL  .MSR
.MSR
```

The message calls used in your program must also be declared in an .MCALL directive; for example,

```
.MCALL  $MDCL,$MREM,.NTCI,.NTCR,.NTLS,.NTLA,.MRCV
```

As implied in the previous example, there are two forms for each message call: the *dot* form and the *dollar* form.

Dot Form of Message Calls

The *dot* form (for example, .NTCI) expands to code that

1. Inserts values that you specify into a 10-word argument block at execution time. You must allocate space for the argument block in a data section of your program. Only the arguments specified in the call are stored in the block. Any unused or unspecified arguments in the argument block are left untouched — denoted throughout this chapter with ///////. The arguments are stored in the block at execution time. Thus, the arguments you specify in the call should be appropriate as operands in MOV instructions.

2. Moves the address of the argument block into R0. (You must save R0 before each message call, if necessary, and restore it later. R0 is set with an error code after each call.)

3. Calls a subroutine named $$$MSR. This subroutine examines the argument block to determine which of the message operations is to be executed. It then sets up the job's File Request Queue Block (FIRQB) and Transmit Request Block (XRB) and executes a .MESAG monitor directive. This directive then transfers control to the appropriate section of NSP code in the RSTS/E monitor.

   (The FIRQB and XRB are two areas within the first 1000 bytes of virtual address space reserved for communications between the RSTS/E monitor, the run-time system, and the user job. A general description of the first 1000 bytes of virtual memory, along with descriptions of the FIRQB and the XRB, can be found in the *RSTS/E System Directives Manual.*)

   NSP performs the function requested, if possible, and transfers control back to the user program at the location following the call. If the call succeeds, the carry bit is clear and R0 is set to zero. If the call fails, the carry bit is set and R0 is set to an error code. Your program can test R0 for numeric values or use error mnemonics resolved from the file ERR.STB at link time (see Section 5.1.3).

NOTE

> Appendix C contains the FIRQB and XRB layouts for each of the message calls. These layouts are provided for program debugging only. You should *not* attempt to set up the FIRQB and XRB and invoke the .MESAG directive yourself. DECnet message operations should be accessed *only* with the calls described in this chapter.

For example, consider the hypothetical call, .DECNTE, whose calling sequence is as follows:

```
.DECNTE area,arg1,arg2,arg3,arg4
```

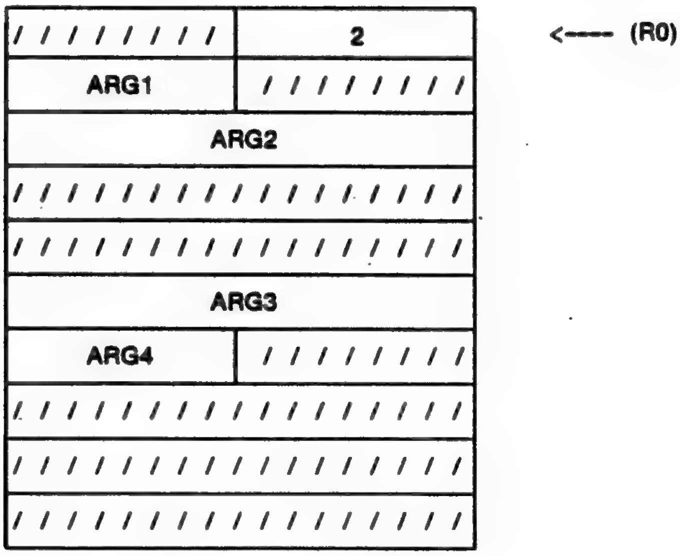
A program might call the macro with code as follows:

```
       .MCALL   .DECNTE
AREA:  .WORD    0,0,0,0,0,0,0,0,0,0     ;DEFINE ARGUMENT BLOCK
          .
          .
          .
       .DECNTE  #AREA,#ARG1,#ARG2,#ARG3,#ARG4
```

After assembly, the code inserted for the .DECNTE macro might look something like the following:

```
MOV             #AREA,R0
MOVB            #2,(R0)
MOVB            #ARG1,3(R0)
MOV             #ARG2,4(R0)
MOV             #ARG3,10.(R0)
MOVB            #ARG4,13.(R0)
JSR             PC,$$$MSR
```

Thus, at execution time, the argument block is filled as follows:



The JSR instruction transfers control to the $$$MSR subroutine that examines the argument block. The first byte defines the operation to be performed and $$$MSR takes it from there.

Dollar Form of Message Calls

The *dollar* form of the message calls (for example, $MDCL) expands to code that simply defines an argument block filled with argument values. The values are filled in at assembly time. Thus, the arguments you specify in the call should be appropriate as operands in MACRO-11 directives such as .BYTE,

.WORD, and so forth. Unlike the *dot* form of the call, any unused or unspecified areas in the argument block are filled with zeros. Since the *dollar* form of the call only allocates and fills the argument block, such calls should be in a data section of your program. To execute the call, your program must load R0 with the address of the argument block, and execute the call to the $$$MSR subroutine.

The *dollar* form of the hypothetical macro discussed previously would then be coded as follows:

```
       .MCALL    $DECNTE
AREA:  $DECNTE   ARG1,ARG2,ARG3,ARG4
          .
          .
          .
       MOV       #AREA,R0
       JSR       PC,$$$MSR
```

After assembly, the code inserted for the $DECNTE macro would be as follows:

```
AREA:  .BYTE   2,0,0
       .BYTE   ARG1
       .WORD   ARG2
       .WORD   0,0
       .WORD   ARG3
       .BYTE   0,ARG4
       .WORD   0,0,0
```

And the argument block created by assembly would be as follows:

0	2	<—— (R0)
ARG1	0	
ARG2		
0		
0		
ARG3		
ARG4	0	
0		
0		
0		

The expansions for the actual message calls are not shown in this chapter. The calls could change or be enhanced in later versions of DECnet/E, and expanding the calls yourself could lead to extensive modifications later. The expansions can, however, be listed in your program by using the .LIST directive.

5.1.2 Assembling

You can assemble MACRO-11 source code that includes DECnet subroutine calls with either one of the two MACRO-11 assemblers available on RSTS/E systems: the MAC assembler (for code to be run under the RSX Run-Time System or its derivatives) or the MACRO assembler (for code to be run under the RT11 Run-Time System).

The library file containing the expansions for the MAC assembler is LB:NETMLB.MLB. This file must be included in the source file list (on the right side of the equal sign) in the assembly. Complete instructions on how to assemble programs using the MAC assembler are given in the *RSTS/E Programmer's Utility Manual.* A typical sequence for an assembly would be as follows:

```
RUN $MAC
MAC>objfile,lstfile=LB:NETMLB.MLB/ML,srcfil1,srcfil2
MAC> ^Z

Ready
```

The library file containing the expansions for the MACRO assembler is $NETMLB.MAC. This file must be included in the source file list (on the right side of the equal sign) in the assembly. Complete instructions on how to assemble programs using the MACRO assembler are given in the *RSTS/E FORTRAN IV Utilities Manual.* A typical sequence for an assembly with MACRO would be as follows:

```
RUN $MACRO
*objfile,lstfile,crefile=$NETMLB.MAC/M,srcfil1,srcfil2
* ^Z

Ready
```

Another useful file to include in the assembly source file list for either MAC or MACRO assemblies is the file COMMON.MAC. DECnet/E calls use the FIRQB and the XRB to return data to the user program. COMMON.MAC assigns the mnemonics FIRQB and XRB to the address of the first byte of the appropriate areas and also contains useful macros. One in particular, .BLKW0, is useful in conjunction with the message calls. Like the MACRO-11 .BLKW directive, .BLKW0 allocates a specified number of words of storage. In addition, .BLKW0 fills the contents of the locations allocated with zeros. For example,

```
CDB: .BLKW0 120.
```

This statement allocates 120 zero-filled words.

5.1.3 Linking

As with assemblers, two linkers are available on RSTS/E systems. Use the Task Builder (TKB) to link object modules produced by the MAC assembler (RSX Run-Time System and its derivatives). The Task Builder will automat-

ically resolve references to the message subroutines from the system library (LB:SYSLIB.OLB). The Task Builder is described in detail in the *RSTS/E Task Builder Reference Manual*. A typical link operation with TKB might be as follows:

```
RUN $TKB
TKB>binfile,mapfile=objfile1,objfile2,...
TKB>Z

Ready
```

Use the LINK routine to link object modules produced by the MACRO assembler (RT11 Run-Time System). LINK resolves references to the message subroutines from the system library ($SYSLIB.OBJ). LINK is described in detail in the *RSTS/E FORTRAN IV Utilities Manual*. A typical link operation with LINK might be as follows:

```
RUN $LINK
*binfile,mapfile=objfile1,objfile2,...
* ^Z

Ready
```

Another useful file that can be included in a link operation is the file ERR.STB. ERR.STB defines mnemonics for the numeric error codes returned in R0 by the message calls. The descriptions in this chapter list the error mnemonics provided by ERR.STB, as well as the octal error values. The symbols will be resolved automatically at link time; you do not need to include ERR.STB in the source file list at link time.

5.2 Declare Receiver (MDCL) — Privileged (Local and DECnet)

The MDCL call must be executed before a program can receive any messages or send any network messages. (Local messages can be sent without first executing this call.) The call defines an identifying object type code or name (or both) and any restrictions on incoming messages. The monitor associates this information with the RSTS/E job number, setting up a Receiver ID Block for the job as described in Section 3.2.3. Only one MDCL call can be in effect at a time for a particular job. It remains in effect until a Remove Receiver (MREM) call is executed for the job.

Macro Format:

.MDC *area,namptr,objtyp,access,bmax,mmax,lmax*

or

label: $MDCL *namptr,objtyp,access,bmax,mmax,lmax*

Argument Descriptions:

label

The label used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MDCL call, the argument block will contain:

/ / / / / / / / / / / / / /	+1
access	objtyp
namptr	
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
lmax	mmax
bmax	
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /

namptr

This word is the address of a six-byte area containing the receiver's logical name. The logical name itself consists of one to six ASCII characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underline). If less than six characters are used, the name must be left-justified in the 6-byte area and the remaining bytes must be filled with spaces. Leading or embedded spaces are invalid. If no name is to be specified (that is, the logical name is null), this area should contain six spaces.

The name is used to identify the calling program for queuing messages from local or network programs. Local programs can address the calling program by the logical name. Network programs can use either the logical name or the object type code, if one is declared.

For local or network programs to access the calling program by logical name, the name must be nonnull and unique to the node. That is, only one program at the node can declare its identity with the name at any given time. NSP uses the logical name to identify the calling program for queuing messages from local or network programs. If the logical name is null, only network access by object type code is allowed. (See the following discussion of *logical name*, *objtyp*, and *access*.)

objtyp

This byte value specifies the object type code, another form of network addressing, As discussed in Section 3.2.1, the object type code defines some service the calling program performs. If the program issuing the MDCL call is addressed by object type code (*name* is null and *access* indicates network access only), multiple copies can execute at the same time in the RSTS/E environment. Unlike the logical name, the object code need not be unique within the node — multiple copies of a program can declare their identity with the same type code simultaneously. If *name* is null, *objtyp* cannot be zero. (Network addressing is discussed in detail in Section 4.1.)

Acceptable values for the object type code range from 0 through 255 (0 to 377 octal). The value 0 indicates that other programs in the network will never address the local program by object type code. The range from 1 to 127 (1 to 177 octal) is reserved for DECnet use (see Appendix A). The range from 128 to 255 (200 to 377 octal) is available for definition and use by a network installation.

access

This byte contains bit flags that are used to determine access to the calling program and to modify certain aspects of the message operations.

The low-order three bits limit the types of senders that can queue messages for the calling program. They do not, however, limit the messages that the calling program can send.

bit 0 = 0 Local senders cannot queue messages for the calling program. (Local senders who use the network functions are considered network senders in this context.)

= 1 Local senders can queue messages for the calling program.

bit 1 = 0 Both privileged and nonprivileged local senders can queue messages for the calling program.

= 1 Only privileged local senders can queue messages for the calling program.

NOTE: This bit is ignored if bit 0 = 0 (that is, if no local senders are allowed).

bit 2 = 0 NSP rejects any incoming requests for logical links.

= 1 NSP queues incoming Connect Initiate Messages for the local program, subject to the declared link maximum (*lmax*) and the declared message maximum (*mmax*). In this case, the user program must provide its own protection from unauthorized access, if desired.

A useful way to document and set these bits is to assign mnemonics to set the bit values:

L = 1
P = 2
N = 4

The byte value can then be set using a statement of the form ACCESS = #L+P+N.

Table 5–1 summarizes access code values and the access permitted for each.

Table 5–1: Types of Access Permitted

Access Code (bits 0,1,2)	Network Logical Links	Local Senders Must Be Privileged	Local Senders Allowed
0	no	no	no
1	no	no	yes
2	no	no	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	no	no
7	yes	yes	yes

Bit 3 of the *access* byte is used to regulate single-link programs, modifying the function of the program's declared link maximum (*lmax*). If bit 3 = 0, incoming requests for logical links do not affect the link maximum. If bit 3 = 1, however, after queuing an incoming Connect Initiate Message to the receiver, NSP sets the program's link maximum to zero, inhibiting all further incoming connect requests. This, in effect, modifies the meaning of the link maximum from "one link at a time" to "one link per program execution."

By setting this "one-shot" bit to 1, a single-link program prevents the possibility that NSP will queue a Connect Initiate Message for it after its one logical link is disconnected and before it issues a Remove Receiver call.

The DECnet/E FAL utility, for example, sets the one-shot bit to 1 and declares a link maximum of 1. Once FAL is started as the result of an incoming request from a remote NFT utility, NSP queues the Connect Initiate Message and zeros the link maximum. Thus, there is no possibility that another incoming connect request could be queued for that copy between the time that NFT breaks the first link and the time that FAL issues its Remove Receiver call. This ensures that each incoming connect request starts a new copy of FAL.

Bit 4 of the *access* byte is used to modify the function of the RSTS/E conditional .SLEEP monitor directive. If bit 4 = 0, any unreceived message queued for the program will block the execution of a conditional Sleep call. This is the normal RSTS/E function. If bit 4 = 1, however, the RSTS/E monitor will not check the program's message queue when determining whether or not it should in fact suspend program execution. Several other conditions (such as a delimiter received on an open terminal) can block the Sleep call, but a pending message will not.

logical name, objtyp, and access

NSP checks the logical name (whose address is given in *namptr*), and the *objtyp* and *access* arguments to ensure that they are consistent. For example, a null logical name and an access allowing local senders are inconsistent, since local senders can address the program by logical name. NSP checks these arguments as shown in Table 5–2. If the arguments are invalid, NSP returns an error (BADFUO) in R0. If the arguments are valid, NSP allocates a small buffer to hold the information passed. If no buffers are available, the call fails with an error (NOROOM). A retry may succeed.

Table 5–2: System Validation of Name, Object, and Access Parameters

Access Code bits 1–2	Senders Permitted	Object Type	Logical Name
0,2	None	Ignored	Ignored.
1,3	Local only	Ignored	Must be nonnull and unique.

(continued on next page)

Table 5-2 (Cont.): System Validation of Name, Object, and Access Parameters

Access Code bits 1-2	Senders Permitted	Object Type	Logical Name
4,6	Network only	Zero	Must be nonnull and unique.
		Nonzero	If *name* is null (at least one leading zero byte), the only access permitted is by object type. Multiple copies of the program can coexist. If *name* is nonnull, it must be unique at the local node. Network senders can refer to the local program by name or by object type. Only one copy of the program using this logical name can execute at any given time.
5,7	Network/Local	Any value	Must be nonnull and unique, so that local senders can refer to the calling program by logical name. Network senders can use the logical name or object type code (if *objtyp* ≠ 0). Only one copy of the program using this logical name can execute at any given time.

bmax

Until a Receive call (MRCV) is executed, a pending message occupies system buffer space. One 16-word buffer from the monitor's buffer pool is used for user-defined or DECnet-defined parameters and other system-specific information. Additional buffer space to hold the message data is usually allocated from the extended buffer pool. If an extended pool does not exist, or no space is available there, the monitor's buffer pool is used.

Because the monitor pool is a critical system resource, the program must set a limit on the amount of space to be used on its behalf. The *bmax* word value sets a limit (1 to 32767 decimal bytes) on the total monitor pool space to be used for the user data portion of messages.

When the number of bytes of monitor pool space used to store message data exceeds the program's declared buffer maximum, NSP no longer queues Local Data Messages for the local program. An error is returned to any local program that tries to send a Local Data Message at this point. Network messages are queued regardless of the buffer maximum but any monitor pool space used is counted against the maximum.

A zero or negative value indicates that the monitor's buffer pool is not to be used for the user data portion of pending Local Data Messages.

mmax

The message maximum byte value (*mmax*) limits the number of messages that can be queued for the calling program. The value can range from 0 to 255 (decimal).

NSP keeps a count of the total number of messages queued for each declared receiver. When one of these counters equals or exceeds the maximum set by the receiver:

1. Incoming Local Data Messages are no longer queued for the receiving program. NSP returns an error to any local sender that tries to send a message when the receiving program's pending message count equals or exceeds *mmax*.
2. Incoming Connect Initiate Messages cause NSP to see if another copy of the program can be started automatically (see Section 4.1.2). If not, NSP rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.
3. An incoming Network Data Message causes NSP to inhibit incoming data messages on that particular logical link and to negatively acknowledge (NAK) the message. This forces the remote system to retransmit when flow is reenabled. See Section 4.2.2 for further details on backpressure flow control.

lmax

The link maximum byte (*lmax*) limits the number of incoming requests for logical links for this program. If the number of links currently active is greater than or equal to *lmax* (0 to 63 decimal) when a remote Connect Initiate Message is received for this program, the local NSP determines if another copy of the program can be started automatically (see Section 4.1.2). If not, NSP rejects the connection and the Connect Initiate Message is not queued. A Connect Reject Message is returned to the sender.

By declaring a small link maximum, a program can avoid the overhead of responding to remote Connect Initiate Messages when it is known beforehand that the program can only handle a limited number of logical links.

A zero value for *lmax* has the same effect as setting bit 2 of the *access* byte equal to zero. That is, NSP rejects all incoming requests for logical links to this program. The *lmax* argument does not, however, limit the number of logical links initiated by this program. That is, it does not stop the program from sending Connect Initiate Messages.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
3	INUSE	The calling job already exists in the system's list of declared receivers. This error may indicate a logic error in the program or that a previous program, running under the same job number, failed to remove itself from the receiver list before terminating. In the latter case, a Remove Receiver call should be issued, followed by another Declare Receiver call. (It is common practice to code a Remove Receiver call immediately before the Declare Receiver call.)
4	NOROOM	There were no small buffers available to hold the arguments passed with the Declare Receiver call. Since the system's use of small buffers is dynamic, a retry may succeed.
10	PRVIOL	The calling program must be privileged at the time the Declare Receiver call is executed.
16	FIEXST	The logical name passed is being used by another job.
18	BADFUO	The parameters passed are inconsistent. See *logical name, object, and access* discussion.

Example:

The following example shows a Declare Receiver call allowing only network access via logical links. Both a logical name (FENWCK) and an object type code (130 decimal) are declared. Since local messages are not allowed, the buffer maximum is set to zero. The message maximum is set to 5. If more than 5 messages accumulate for this program, an incoming request for a connection causes NSP to determine if another copy of the program can be started automatically. If not, NSP rejects the link. Any incoming data messages at this point also cause NSP to inhibit incoming message flow on all the program's logical links because of the backpressure condition. The link maximum is set to two. When the program has established two logical links, an incoming Connect Initiate Message will cause NSP to check to see if another copy of the program can be started. If not, NSP will reject the link.

```
AREA:   .BLKWO 10.
LNAME:  .ASCII /FENWCK/
          .
          .
          .
        .MDCL   #AREA,#LNAME,#130.,#4,#0,#5,#2
```

The following example shows the $MDCL form of the same call:

```
LNAME:  .ASCII /FENWCK/
AREA:   $MDCL  LNAME,130.,,4,0,5,2
                .
                .
                .
        MOV    #AREA,R0
        JSR    PC,$$$MSR
```

bufoff

This word defines an offset, in bytes, from the address specified in *bufptr*. The value of *bufoff* is added to *bufptr* to define the beginning address of the returned data.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes of data would be processed, starting with the byte at *bufptr*-100.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
4	NOROOM	The event queue is full or there are no small buffers available to hold the event. A retry may succeed. (This error is returned only if *evmod* = 0.)
5	NOSUCH	The event logger is not running. Have the system manager start the event logger.
10	PRVIOL	The calling program must be privileged at the time the NTEV call is issued.
18	BADFUO	The parameters passed are either inconsistent or invalid.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example illustrates the logging of a user event, class 480. and type 3. The entity is node number 135. The optional data passed for processing consists of an *lla* parameter number and value. A "missed event" is logged if the event itself cannot be.

```
ARGBLK: .BLKW   10              ;DEFINE NTEV ARGUMENT BLOCK

MP$LLA  = 2130
ME$NOD  = 0
BUFOFF  = 6

EVBUF:  .BLKW   6
EVPARM: .WORD   MP$LLA          ;STORAGE FOR EVENT DATA
EVLEN:  .BYTE   2
EVVAL:  .BLKB   2

        .
        .
        .
MOV LLA,RO                      ;GET LLA NUMBER
MOVB    RO,EVVAL                ;STORE IN BUFFER
SWAB    RO
MOVB    RO,EVVAL+1
.NTEV   #ARGBLK,#1,#480.,#3,#ME$NOD,#135.,#EVBUF,#5,#BUFOFF
```

The following example shows the $NTEV form of the same call.

```
ARGBLK: $NTEV  1,480.,3,ME$NOD,135.,EVBUF,5,BUFOFF

MP$LLA   = 2130
ME$NOD   = 0
BUFOFF   = 6

EVBUF:   .BLKW  6
EVPARM:  .WORD  MP$LLA      ;STORAGE FOR EVENT DATA
EVLEN:   .BYTE  2
EVVAL:   .BLKB  2

         .
         .
         .
MOV      LLA,R0             ;GET LLA NUMBER
MOVB     R0,EVVAL           ;STORE IN BUFFER
SWAB     R0
MOVB     R0,EVVAL+1
MOV      #ARGBLK,R0         ;DO IT
JSR      PC,$$$MSR
```

The system calls in both the previous examples generate event messages in the following format:

```
Event type 480.3
Occurred 14-Dec-81 15:31:39.7 on node 135 (GROK)
Node address 135 (GROK)
Parameter #2130 = 40960
```

5.6 Send Calls

There are nine separate send calls — one local and eight network. Each of these calls causes NSP to format a message that is then either queued to a local program or transmitted across the network to a remote program.

5.6.1 Send Local Data Message (MSLD) (Local)

The MSLD call sends up to 532 (decimal) bytes of user data to another program on the local RSTS/E system. The other program must be a declared receiver of local messages. If the other program has declared that only privileged local senders are allowed, this program must be privileged. The receiving program is identified in the MSLD call either by its job number or by the logical name with which it declared itself a receiver.

Macro Format:

.MSLD *area,jobx2,namptr,bufptr,buflen,bufoff,parptr,parlen*

or

label: $MSLD *jobx2,namptr,bufptr,buflen,bufoff,parptr,parlen*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MSLD call, the argument block will contain the following:

jobx2	-1
namptr	
bufptr	
buflen	
bufoff	
parptr	
parlen	
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /

jobx2

If *jobx2* = 0, the message data is sent to the local program identified by the logical name whose beginning address is specified in *namptr*. Otherwise, the message data is sent to the program whose job number (times two) is equal to the value of *jobx2*. For example, if *jobx2* = 8, the message is sent to job 4.

namptr

This word is the address of a 6-byte area containing the logical name of the local program to which the data is to be sent. The logical name itself consists of one to six ASCII characters. Valid characters are uppercase alphanumerics and the special characters "$" (dollar sign), "." (period), and "_" (underline). If the name is less than six characters, it must be left-justified in the 6-byte area and the remaining bytes must be filled with spaces. Leading or embedded spaces are invalid. This argument is ignored if *jobx2* ≠ 0.

bufptr

This word defines the starting address of the message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *bufptr*.

buflen

This word defines the number of bytes to be sent from the *bufptr* buffer. Possible values for *buflen* range from 0 through 512 (decimal).

bufoff

This word defines an offset, in bytes, from the address specified in *bufptr*. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the message data to be sent.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 512 and a *bufoff* of 1000. The MSLD call would send 512 bytes beginning at *bufptr*+1000.

parptr

This word is the starting address of an area containing up to 20 (decimal) bytes of parameter message data to be sent to the local program. This data will be delivered separate from the message data in the *bufptr* area. For example, a MACRO program receiving a Local Data Message will get the parameter message data in the FIRQB and the other data in a buffer defined in the Receive call.

parlen

This word defines the number of bytes to be sent from the parameter area whose starting address is defined by *parptr*. The *parlen* value can range from 0 through 20 (decimal).

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
4	NOROOM	The number of pending messages for the intended local receiver is at its declared maximum. The sending program should try again later. If this error occurs repeatedly, the receiver is not processing messages often enough.
5	NOSUCH	The intended local receiver could not be located in the system's list of declared receivers. The receiver must be declared (with a Declare Receiver call) before any data can be transmitted to it.
10	PRVIOL	Some access violation has occurred. Either the receiver does not allow any local senders or the sender is nonprivileged and the receiver allows only privileged senders.
18	BADFUO	The value of *jobx2* is odd. It must be 0 or the receiver's job number times two.
31	BADCNT	Either the *buflen* value or the *parlen* value is invalid. The *buflen* argument can range from 0 through 512 (decimal). The *parlen* argument can range from 0 through 20 (decimal).
32	NOBUFS	System buffers are currently not available to store this message for the intended local receiver. A later retry may proceed without error.

Example:

The following example shows a local send to job number 23. A buffer length of 6 and offset of 18 indicate that the characters "OFFSET" are sent.

```
ARGBLK: .BLKWO 10.
BUFFER: .ASCII /TO ILLUSTRATE THE OFFSET/
          .
          .
          .
        .MSLD   #ARGBLK,#46.,,#BUFFER,#6.,#18.,#0,#0
```

The following example shows the $MSLD form of the same call:

```
ARGBLK: $MSLD   46.,,BUFFER,6.,18.
          .
          .
          .
        MOV     #ARGBLK,RO
        JSR     PC,$$$MSR
```

5.6.2 Send Connect Initiate Message (NTCI) (DECnet)

The NTCI call requests a logical link to another program. The call establishes the user link address that the calling program will use to refer to the link. The other program is identified with a Connect Data Block. Other arguments specify the flow control option and maximum amount of user data for received Network Data Messages. Up to 16 (decimal) bytes of user data can be sent with the NTCI call.

Macro Format:

.NTCI *area,ula,llmod,rmax,bufptr,buflen,bufoff*

or

label: $NTCI *ula,llmod,rmax,bufptr,buflen,bufoff*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTCI call, the argument block will contain:

ula	-2
/ / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / /
bufptr	
buflen	
bufoff	
/ / / / / / / / / / / / / /	llmod
rmax	
/ / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / /

ula

The user link address (*ula*) is a byte value in the range of 1 to 255 (1 to 377 octal). Later calls to send and receive messages use this value to refer to the logical link (assuming that the link being requested by the NTCI is accepted by the remote NSP and the remote program).

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

bufptr

This word defines the starting address of the Connect Data Block and optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

The Connect Data Block identifies the target program for the requested logical link. Figure 5–1 shows the format of the Connect Data Block. The fields within the block are described in Table 5–3. Up to 16 (decimal) bytes of user data can follow the Connect Data Block.

buflen

This word defines the number of bytes to be sent. Since the Connect Data Block is 120 (decimal) bytes long and the optional user data can be from 0 to 16 (decimal) bytes long, acceptable values for *buflen* range from 120 through 136 (decimal).

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the Connect Data Block.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 120 and a *bufoff* of -100. The beginning address of the Connect Data Block would then be interpreted as (*bufptr*-100).

llmod

The value of the *llmod* byte indicates the type of flow control requested by the calling program to control incoming data from the remote program. Acceptable values and meanings are:

0 No flow control
1 Segment flow control
2 Message flow control

Flow control is described in detail in Chapter 4. Remember that both programs on a logical link select their own flow control independent of each other. The *llmod* value here is the selection made by the program issuing the call. If *llmod* is nonzero, the calling program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

rmax

A program could have limited buffer space and not be coded to process large messages in small pieces. Such a program can impose a limit on the amount of user message data it is willing to receive on a logical link. The word value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by NSP will itself limit the amount of data that can be passed in a single Network Data Message. If the maximum size set for NSP by network management is not large enough to handle the receive maximum as specified in *rmax*, the local NSP will alter the maximum to the smaller limit before forwarding the Connect Initiate Message to the remote node. If *rmax* = 0, NSP will supply the size of its receive buffers as a default. (The calling program will be informed of any such change by a field in the Connect Confirm Message from the target program when the link is accepted.)

The remote NSP uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument for the Receive call (MRCV) in Section 5.7 for a description of how to process large messages in small pieces.

Decimal Position	Octal Offset			Octal Offset	Decimal Position
		REMOTE NODE NAME IN ASCII (SPACE FILL TO 6 BYTES)		0	1
6	5			4	5
8	7	OBJECT TYPE	FORMAT	6	7
10	11	DESCRIPTOR LENGTH	0	10	9
12	13	REMOTE DESCRIPTOR		12	11
• • •	• • •				
22	25			24	21
24	27	GROUP CODE FOR FORMAT 2		26	23
26	31	USER CODE FOR FORMAT 2		30	25
28	33	OBJECT TYPE	FORMAT	32	27
30	35	DESCRIPTOR LENGTH	0	34	29
		LOCAL DESCRIPTOR		36	31
• • •	• • •			• • •	• • •
44	53	GROUP CODE FOR FORMAT 2		52	43
46	55	USER CODE FOR FORMAT 2		54	45
48	57	ID LENGTH	0	56	47
50	61	USER IDENTIFICATION		60	49
• • •	• • •			• • •	• • •
64	77			76	63
66	101	PASSWORD LENGTH	0	100	65
68	103	PASSWORD		102	67
74	111			110	73
76	113	ACCOUNT LENGTH	0	112	75
78	115	ACCOUNTING INFORMATION		114	77
92	133			132	91
94	135	RESERVED — MUST BE FILLED WITH ZEROS		134	93
120	167			166	119

Filled in by local NSP for Send Connect Initiate Message

Figure 5–1: Format of Connect Data Block

Table 5-3: Format of Connect Data

Octal Offset	Content
0-5	**Remote Node.** 1 to 6 uppercase, alphanumeric ASCII characters naming the remote node to which the Connect Initiate Message is to be sent. The name must contain at least one alphabetic character, and names shorter than six characters must be left-justified and padded to six characters with spaces. If no node name is to be specified (that is, the request is directed to a program at the local node), these 6 bytes should be spaces.
6-31	**Remote Program.** Identifies the remote program to which the Connect Initiate Message is directed by either an object type code (general function) or a specific entity (program, task, process). This identification is given in one of three formats:

Format 0

The remote program is identified by object type code alone. The fields are:

Octal Offset	Content
6	Zero, to indicate a format 0 name.
7	Object type code of the remote program.
10-31	Zero (not used).

Format 1

The remote program is identified by a descriptor. The fields are:

Octal Offset	Content
6	One, to indicate a format 1 name.
7	Normally zero. This byte should be zero unless the target DECnet system allows identification by both name and object type code on incoming connect requests (DECnet/E does not). If nonzero, this byte is interpreted as the object type code of the remote program for which the Connect Initiate Message is intended.
10	Reserved - must be zero.
11	Length of descriptor, 0 to 16 (decimal) bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (offset 12-31 octal).
12-31	Remote program descriptor (left-justified). For a DECnet/E program this is the logical name that the remote program defined in its Declare Receiver call.

Format 2

This format allows identification of the remote program by name or object type code and by the group and user codes under which the remote program is accessible on the remote system. (The group and user codes refer to what is called the project-programmer number on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use only the object type code or descriptor to find or start the target program. The group and user codes are simply passed on to the target program for its own checking. Other DECnet systems may handle an incoming connect request in format 2 differently.

(continued on next page)

Table 5-3 (Cont.): Format of Connect Data

The subfields are defined as follows:

Octal Offset	Content
6	Two, to indicate a format 2 name.
7	Object type code of the remote program, if applicable. If zero, the descriptor must be used.
10	Reserved - must be zero.
11	Length of descriptor, 0 to 12 (decimal) bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (octal offset 12-25).
12-25	Remote program descriptor, if this is used instead of object type code. The descriptor should be left-justified in the field.
26-27	Remote group code. Binary value giving the group number under which the remote program is running or is to be started. Corresponds to the project number in a RSTS/E project-programmer number. This value is not used by DECnet/E systems but is simply passed on to the receiving program.
30-31	Remote user code. Binary value giving the user number under which the remote program is running or is to be started. Corresponds to the programmer number in a RSTS/E project-programmer number. This value is not used by DECnet/E systems but is simply passed on to the receiving program.

32-55 **Local Program**. This field is filled in by the local NSP using information that the local program specified in its Declare Receiver call. Any values passed by the user program in this area are ignored.

The information is passed to the remote system in format 2, as described previously. Byte offset 32 is filled in with a value of 2. If the program declared its identity with an object type code alone, the object type code is passed in byte 33 and bytes 36-55 are zero. If the program declared its identity with a logical name, or with a logical name and object type code, the local name is supplied in octal offset bytes 36-43 and octal offset byte 33 is zero. The length of the logical name is given in octal offset byte 35. The project-programmer number under which the program is running is passed in octal offset bytes 52-55 as the group and user codes. The project number is in byte 52. The programmer number is in byte 54. Both are in binary format.

56-133 **Access Control Fields**. These bytes can be used to define the calling program's access rights to the remote system or to the remote system's services. The idea is analogous to logging in on the remote system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number. Thus, three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system simply passes this information on to the receiving program. To determine what, if anything, a non-RSTS/E DECnet system requires in these fields, see the DECnet manue' for that system.

(continued on next page)

Table 5–3 (Cont.): Format of Connect Data

NOTE

The DECnet/E utilities NFT and NETCPY, which use logical links to access remote files and devices, use these fields to establish the local terminal user's right to access the remote files and devices. Both utilities prompt the local user for log-in information to be used at the remote system and pass that information along in the Connect Data Block of the Connect Initiate Message. The remote FAL or receiving NETCPY checks this information before allowing the operation requested.

Octal Offset	Content
56	Reserved – must be zero.
57	Length of user identification information specified in the next field, 0 to 16 (decimal) bytes.
60–77	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the project-programmer number used at log-in. This information need only be specified here if it is required by the remote system or program. If it is not required, the length (octal byte offset 57) should be set to zero.
100	Reserved – must be zero.
101	Length of password information specified in the next field, 0 to 8 (decimal) bytes.
102–121	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the project-programmer number at log-in. A password supplied here must be acceptable to the remote system or program, if one is required. If one is not required, the length (octal byte offset 101) should be set to zero.
122	Reserved – must be zero.
123	Length of the accounting information specified in the following field, 0 to 16 (decimal) bytes.
124–143	Accounting information. Many systems require a billing account number in addition to user identification and password. The accounting field is provided for this purpose. Once again, such information need only be supplied here if it is required by the remote system or program. If it is not required, the length (octal byte offset 123) should be set to zero.

134–167 **Reserved for future use**. Currently ignored by DECnet/E but should be passed as zeros.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
3	INUSE	The calling program already has a logical link with the same user link address (*ula*) as specified in this Connect Initiate Message. A different *ula* must be used or the existing link must be disconnected.

(continued on next page)

Decimal Value	ERR.STB Mnemonic	Meaning
6	NODEVC	The node named in the Connect Data Block is not known to the system.
14	HNGDEV	The node named in the Connect Data Block is known to the system but is currently inactive. There is no physical communication path to the node.
17	DTOOOF	NSP cannot allocate its local link address (*lla*) for the logical link. The maximum number of logical links allowed at the local RSTS/E node (63 or some limit between 1 and 63 — set by the system manager) has been reached. A later try may succeed. (See the system manager if the limit in effect is inadequate.)
18	BADFUO	One of several possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The Declare Receiver call for the calling program had a null logical name and a zero object type code. There is no way to identify the calling program to the remote system. Issue a Remove Receiver call, followed by another Declare Receiver call. 3. An invalid value was detected in one of the following fields. • Remote program field in the Connect Data Block. • One of the three access control fields in the Connect Data Block. • The user link address (*ula*) given was zero. Valid values range from 1 to 255 (1 to 377 octal). • The local link modifier (*llmod*) value must be 0, 1, or 2. • The remote descriptor length (octal byte offset 11 in the Connect Data Block) was nonzero but the descriptor field was null (all zeros).
22	PAKLCK	NSP cannot create a new logical link because the physical link to the remote node or the local node itself has been scheduled for shutdown by the system manager.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 120 and 136 (decimal) bytes for a Connect Initiate Message.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet cannot be executed until the system manager enables NSP.
68	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

This example shows a NTCI call used to direct a Connect Initiate Message to a program with object type code 140 at node MIAMI. The Connect Data Block uses a format 0 name to address the remote program. The local program name

is supplied by NSP and no access control fields are being used. Thus, the rest of the Connect Data Block is filled with zeros to make up 120 bytes.

The user link address is 15 (decimal), segment flow control is requested, and the receive maximum is 512 (decimal) bytes.

```
ARGBLK: .BLKW0 10.        ;DEFINE ARGUMENT BLOCK
CDB:    .ASCII /MIAMI /
        .BYTE  0,140.
        .BLKW0 114.
            .
            .
            .
        .NTCI  #ARGBLK,#15.,#0,#512.,#CDB,#120.,#0
```

The following example shows the $NTCI form of the same call.

```
ARGBLK: $NTCI  15.,0,512.,CDB,120.,0
CDB:    .ASCII /MIAMI /
        .BYTE  0,140.
        .BLKW0 114.
            .
            .
            .
        MOV    #ARGBLK,R0
        JSR    PC,$$$MSR
```

5.6.3 Send Connect Confirm Message (NTCC) (DECnet)

The NTCC call accepts a remote program's request for a logical link. The call establishes the user link address that the calling program will use to refer to the link. The local link address assigned by the local NSP identifies the particular link being accepted. The desired flow control option and the receive maximum for the link are specified, and up to 16 (decimal) bytes of message data can be passed to the remote program.

Macro Format:

.NTCC *area,ula,lla,llmod,rmax,bufptr,buflen,bufoff*

or

label: $NTCC *ula,lla,llmod,rmax,bufptr,buflen,bufoff*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTCC call, the argument block will contain:

ula	-3
lla	
bufptr	
buflen	
bufoff	
/ / / / / / / / / / / / / /	llmod
rmax	
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /

ula

The user link address (*ula*) is a byte value in the range of 1 to 255 (1 to 377 octal). Later calls to send and receive messages use this value to refer to the logical link.

The *ula* must be unique within the program at any given time. That is, one program cannot have two different logical links with the same *ula* at the same time.

lla

The local link address (*lla*) is a word value that identifies the link being accepted. It is a number assigned by the local NSP when it received the Connect Initiate Message requesting the link. The local link address is passed to the local program in the FIRQB at octal offset 6 when the program receives a Connect Initiate Message. (Section 5.7.2 describes the information returned for a received Connect Initiate Message.)

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent: 0 to 16 (decimal).

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user data.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100 would be sent.

llmod

The value of the *llmod* byte indicates the type of flow control requested for this end of the logical link. Acceptable values and meanings are as follows:

- 0 No flow control
- 1 Segment flow control
- 2 Message flow control

Flow control is described in detail in Chapter 4. Remember that both programs in a logical link select their own flow control independent of each other. The *llmod* value here is the selection made by the local program. If *llmod* is nonzero, the program must issue Link Service (NTLS) calls to request Network Data Messages from the other program.

A program could have limited buffer space and not be coded to process large messages in small pieces. Such a program can impose a limit on the amount of user message data it is willing to receive on a logical link. The word value *rmax* specifies this limit in bytes.

The size of the receive buffers allocated by NSP will itself limit the amount of data that can be passed in a single Network Data Message. If the maximum size set for NSP by network management is not large enough to handle the receive maximum as specified in *rmax*, the local NSP will alter the maximum to the smaller limit before forwarding the Connect Confirm Message to the remote node. If *rmax* = 0, NSP will supply the size of its receive buffers as a default value.

The remote NSP uses this value to establish a transmit maximum for its end of the logical link. If the remote system is also a RSTS/E system, this transmit maximum is passed on to the remote program. The remote program must limit the amount of user message data it sends over the logical link according to this value.

See the discussion of the *buflen* argument for the Receive call (MRCV) in Section 5.7 for a description of how to process large messages in small pieces.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
3	INUSE	A logical link is currently active for the calling program with the same user link address (*ula*) as specified in this Connect Confirm Message. A different *ula* must be used or the existing link must be disconnected.
5	NOSUCH	The local link address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the latter case, a Link Abort Message has already been queued for this link. (The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established).
10	PRVIOL	Some procedural error has occurred. Sending a Connect Confirm Message for a link that is not waiting for confirmation will cause this error. For example, a second Connect Confirm Message for the same logical link will cause this error because the link is already established.

(continued on next page)

Decimal Value	ERR.STB Mnemonic	Meaning
18	BADFUO	One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. An invalid value was detected in one of the following fields. • The user link address (*ula*) given was zero. Valid values range from 1 to 255 (1 to 377 octal). • The local link modifier (*llmod*) value must be 0, 1, or 2.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 (decimal) bytes for a Connect Confirm Message.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

In this example, a NTCC call responds to a received Connect Initiate Message. No intervening calls to the monitor have been made that would have destroyed the *lla* in the FIRQB so this argument is specified as FIRQB+6. The call also establishes a user link address of 15 (decimal), a receive maximum of 256 (decimal) bytes, and indicates that the calling program requests message flow control. No user message data is sent so, to ensure that no other arguments remain in the argument block from a previous call, these arguments are zero.

```
ARGBLK: .BLKWO  10.         ;DEFINE ARGUMENT BLOCK
        .
        .
        .
        .NTCC   #ARGBLK,#15.,@#FIRQB+6,256.,#0,#0,#0
```

The following example shows the *dollar* form of the same call.

```
ARGBLK: $NTCC   15.,,256.
        .
        .
        .
        MOV     #ARGBLK,R0
        MOV     @#FIRQB+6,2(R0)
        JSR     PC,$$$MSR
```

Note that with the *dollar* form, the *lla* value must be moved from FIRQB+6 to the appropriate place in the argument block at execution time. The *lla* value is not known at assembly time when the argument block is generated by the *dollar* form.

5.6.4 Send Connect Reject Message (NTCR) (DECnet)

The NTCR call rejects a remote program's request for a logical link. The link is identified by the local link address established by the local NSP. Up to 16 (decimal) bytes of user message data can be sent to the remote program as part of the Connect Reject Message.

Macro Format:

.NTCR *area,lla,reason,bufptr,buflen,bufoff*

or

label: $NTCR *lla,reason,bufptr,buflen,bufoff*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTCR call, the argument block will contain the following:

/ / / / / / / / / / / / / /	-4
lla	
bufptr	
buflen	
bufoff	
reason	
/ /	
/ /	
/ /	
/ /	

lla

The local link address (*lla*) is a word value that identifies the link being rejected. It is a number assigned by the local NSP when it received the Connect Initiate Message requesting the link. The local link address is passed to the local program in the FIRQB at octal offset 6 when the program receives a Connect Initiate Message. (Section 5.7.2 describes the information returned for a received Connect Initiate Message.)

reason

DECnet/E NSP allows one of three reason code values for this word: 0, 34 (decimal), or 36 (decimal).

Most user programs will use 0. If any reason is supplied for rejecting the connection, it is passed in the 16-byte user data area.

The values 34 and 36 are allowed so that DECnet/E support programs (such as FAL and NFT) can reject unauthorized or improper requests for access to local RSTS/E files in a manner agreed upon as part of the overall DECnet design. There is no reason for a user program to use these codes unless it talks to other programs that interpret them. All DECnet programs that are concerned with protection against unauthorized access are designed to recognize these codes. These programs take appropriate action (such as displaying an error message to a user) when they receive such a Connect Reject Message.

The value 34 indicates that the user ID and password subfields in the Connect Data Block of the remote program's Connect Initiate Message do not correspond to any valid user known to the local system.

The value 36 indicates that, while the user ID and password subfields are acceptable, the account subfield is not. For example, the specified user may not be authorized to use that billing account or the account may be exhausted.

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent: 0 to 16 (decimal).

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

NOTE

The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100 would be sent.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
5	NOSUCH	The local link address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the latter case, a Link Abort Message has already been queued for this link. (The *lla* field in the received Link Abort Message identifies the link since no *ula* was ever established.)
10	PRVIOL	Some procedural error has occurred. Sending a Connect Reject Message for a link that is not waiting for confirmation will cause this error. For example, sending a Connect Reject Message for a link that is already established will return this error.
18	BADFUO	One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The reason code supplied with the Connect Reject Message was not 0, 34 (decimal), or 36 (decimal).
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 bytes for a Connect Reject Message.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

In this example, a NTCR call responds to a received Connect Initiate Message. No intervening calls to the monitor have been made that would have destroyed the *lla* in the FIRQB so this argument is specified as FIRQB+6. A reason code of 0 is supplied. The local program notifies the remote program of the reason for rejection with user data.

```
ARGBLK: .BLKWO 10.          ;DEFINE ARGUMENT BLOCK
REASON: .ASCII /TOO LATE/   ;DEFINE MESSAGE DATA
                .
                .
                .
        .NTCR   #ARGBLK,@#FIRQB+6,#0,#REASON,#8.,#0
```

5.3 Remove Receiver (MREM) (Local and DECnet)

The MREM call removes the specified job from the system's list of declared receivers. All pending messages are discarded and any active logical links are aborted. This call should be executed at the completion of message processing. This prevents unwanted messages from accumulating in the queue of pending messages.

Privileged programs can remove other RSTS/E jobs with this call, although normally only system utility programs would do this.

Macro Format:

.MREM *area,jobx2*

or

label: $MREM *jobx2*

Argument Descriptions:

label

The label used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MREM call, the argument block will contain:

jobx2	0
/ /	
/ /	
/ /	
/ /	
/ /	
/ /	
/ /	
/ /	

evcls

This word parameter specifies the class of the user event to be logged. Events are divided into classes, according to the DNA layer from which they originate. Classes 480 to 511 are reserved for customer-specific events. However, DECnet/E supports only class 480. Thus, *evcls* must be 480.

evtyp

This parameter specifies the type of event within the class specified by *evcls*. The event type itself is only a 5-bit value; thus, *evtyp* must be in the range of 0 to 31 (decimal).

entyp

This parameter specifies the entity type with which the event is concerned. There are four entity types supported by DECnet/E, defined as follows:

entyp	
-1	No specific entity
0	Node
1	Line
3	Circuit

enval

This parameter associates a value to the entity specified by the *entyp* parameter. If the entity is a node, *enval* should be set to the node address. If the entity is a line or a circuit, *enval* should be set to the address of the Device Data Block (DDB) maintained for the device.

bufptr

This word defines the starting address of the buffer containing the optional user data to be processed with the event. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address specified by *bufptr*.

If this data is included, it must be in NICE protocol format. Basically, this means that the data should be formatted in a 3-part field: a parameter type, a data type, and a parameter value. DECnet parameter types can be used if the event concerns a node, a circuit, or a line. In this case, the event logger will output standard descriptive text for the specific parameter. If a DECnet parameter type is not used, parameter types in the range of 3900 to 4095 (decimal) can be specified. (See the *DECnet DIGITAL Network Architecture, Network Management Functional Specification*, Version 3.0.0, for further information on event definitions and parameters.)

Up to 200 (decimal) bytes of data can be passed to the event logger for processing.

buflen

This word defines the number of bytes of data in the buffer specified with *bufptr*. Up to 200 (decimal) bytes of data can be passed. If no data is specified, *buflen* must be 0.

The following example shows the *dollar* form of the same call.

```
ARGBLK: $NTCR   ,,,REASON,8.
REASON: .ASCII /TOO LATE/
          .
          .
          .
        MOV     #ARGBLK,R0
        MOV     @#FIRQB+6,2(R0)
        JSR     PC,$$$MSR
```

Note that with the *dollar* form, the *lla* value must be moved from FIRQB+6 to the appropriate place in the argument block at execution time. The *lla* is not known at assembly time when the *dollar* form generates the values for the argument block.

5.6.5 Send Network Data Message (NTDM) (DECnet)

The NTDM call transmits user data — a *segment* as defined in Chapter 4 — over an established logical link. A flag byte indicates whether or not the segment is the end of a logical message. If the remote program requested flow control when the link was established, requests for data must be outstanding or this call results in an error.

Macro Format:

.NTDM *area,ula,dmflgs,bufptr,buflen,bufoff*

or

label: $NTDM *ula,dmflgs,bufptr,buflen,bufoff*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTDM call, the argument block will contain:

ula	-5
///////////////	dmflgs
bufptr	
buflen	
bufoff	
///////////////////////////////	
///////////////////////////////	
///////////////////////////////	
///////////////////////////////	
///////////////////////////////	

ula

The *ula* is a byte value specifying the user link address (1 to 255 decimal) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

dmflgs

The low-order two bits of the *dmflgs* byte indicate whether the segment being sent is the first, last, middle, or sole segment of a logical message. If message flow control was requested by the remote program when the link was established, NSP uses these two bits to regulate the transmission of logical messages as the remote program requests them. (See Chapter 4 for a full discussion of flow control.)

Bit 1	Bit 0	
0	0	Middle segment of a logical message.
0	1	First segment of a logical message.
1	0	Last segment of a logical message.
1	1	Sole segment of a logical message.

DECnet/E NSP uses bit 1 to keep its count of logical messages transmitted to the remote program. Bit 0 is not used in maintaining the logical message request count but is simply passed to the remote system and program.

NSP also uses bit 1 to determine whether it will permit a disconnect of the logical link. If a disconnect is requested but the last Network Data Message sent did not have bit 1 set (=1), NSP assumes that the last message has not been completely sent and will not permit the disconnect. This is always done, irrespective of the flow control option selected by the receiver.

Some DECnet implementations require that bit 1 be set to terminate an asynchronous I/O operation. Thus, when the remote program requests segment flow control or no flow control, bit 1 should always be set. Setting bit 1 in this case will not affect a remote DECnet/E node but is essential if the remote node is a DECnet node waiting to terminate asynchronous I/O.

When the sign bit (bit 7) of the *dmflgs* byte is set to 1, link status information is returned to the FIRQB when the NTDM call completes successfully. If link status information is requested, it is returned in the same format as a received Link Service Message (see Section 5.7.7).

A useful way to document and set these flags is shown below:

```
EOM = 2
BOM = 1
RLS = 128.
```

Then, to indicate the end of a logical message and to request the return of link status information, the *dmflgs* argument in the call could be specified as #EOM+RLS.

bufptr

This word defines the starting address of the user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes to be sent. The maximum amount of data that can be transferred over the logical link is established by the remote program or the remote NSP. It is passed on to the calling program as a transmit maximum (the *tmax* value in a received Connect Initiate or Connect Confirm Message, see Sections 5.7.2 and 5.7.3). Thus, the value of *buflen* can range from 1 through *tmax*.

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. One hundred bytes beginning at *bufptr*+20 would be sent.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
4	NOROOM	The Data Message transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Network Data Messages. No more can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should simply retry the send after a short (1 to 5 second) delay. (See Chapter 4 for a full discussion of flow control.)
5	NOSUCH	The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending a Network Data Message over a logical link that has not yet been confirmed will cause this error.

(continued on next page)

Decimal Value	ERR.STB Mnemonic	Meaning
18	BADFUO	One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The *ula* specified is zero. It must be within the range 1 to 255 (decimal).
19	INTLCK	Either the remote system has inhibited transmission on this link because fo a backpressure condition, or there are no outstanding requests for message data from the remote program (assuming it requested segment or message flow control for the link). The calling program will be notified with a Link Service Message when the condition clears, as described in Chapter 4.
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 1 and the transmit maximum (*tmax*) established by the remote program or system in its Connect Initiate or Connect Confirm Message for this logical link.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example shows a 100-byte send from a buffer area. The user link address is 20 (decimal). The value of 203 for *dmflgs* identifies the message data as the sole segment of a logical message and requests the return of link status information.

```
ARGBLK: .BLKWO 10.        ;DEFINE ARGUMENT BLOCK
BUFFER: .BLKWO 100.       ;DEFINE BUFFER FOR SEND
          .
          .
          .
(fill buffer with data to be sent)
          .
          .
          .
        .NTDM   #ARGBLK,#20.,#203,#BUFFER,#100.,#0
```

The following example shows the *dollar* form of the same call.

```
ARGBLK: $NTDM   20.,203,BUFFER,100.
BUFFER: .BLKWO  100.
          .
          .
          .
(fill buffer with data to be sent)
          .
          .
          .
        MOV     #ARGBLK, R0
        JSR     PC,$$$MSR
```

5.6.6 Send Interrupt Message (NTIN) (DECnet)

The NTIN call transmits an Interrupt Message to a remote program over an established logical link. All DECnet systems are designed to deliver Interrupt Messages ahead of other messages. If the remote system is a DECnet/E system, the Interrupt Message will be placed at the head of the pending message queue, behind the first message (since it may already be partly received) and behind any other pending Interrupt Messages queued for the program. Interrupt messages are subject to flow control as described in Chapter 4.

Macro Format:

.NTIN *area,ula,inflgs,bufptr,buflen,bufoff*

or

label: $NTIN *ula,inflgs,bufptr,buflen,bufoff*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTIN call, the argument block will contain the following:

ula	-6
/ / / / / / / / / / / / / /	inflgs
bufptr	
buflen	
bufoff	
/ /	
/ /	
/ /	
/ /	
/ /	

ula

The *ula* is a byte value specifying the user link address (1 to 255 decimal) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

inflgs

If the sign bit (bit 7) of the *inflgs* byte is set to 1, link status information is returned to the FIRQB when the NTIN call completes successfully. If link status information is requested, it is returned in the same format as a received Link Service Message (see Section 5.7.7).

A useful way to document and set this flag is to assign a mnemonic to the value 128 (decimal) to set or clear the flag, as follows:

RLS = 128

Then, to request the return of link status information, the *inflgs* argument can be specified as #RLS.

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent: 0 to 16 (decimal).

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address user message data.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100 would be sent.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
4	NOROOM	The Interrupt/Link Service transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and no Link Service Message notification occurs when the condition clears. The program should simply retry the send after a short (1 to 5 second) delay. (See Chapter 4 for a full discussion of flow control.)

(continued on next page)

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	The local link address (*lla*) does not correspond to any known logical link for the calling program. Either the *lla* is incorrect or the originating connect request has been cancelled. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending an Interrupt Message over a logical link that has not yet been confirmed will cause this error.
18	BADFUO	One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The user link address (*ula*) given was zero. Valid values range from 1 to 255 (1 to 377 octal).
19	INTLCK	The remote NSP has prohibited transmission of Interrupt Messages over this logical link according to its rules for Interrupt Message flow control. A Link Service Message will be delivered to the calling program when the condition clears. (See Chapter 4 for a full discussion of flow control.)
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 (decimal) bytes for an Interrupt Message.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example shows an NTIN call. The user data "TIMESUP" is passed to a remote program over logical link 15. (The remote program would have to be coded to recognize this message and take appropriate action.)

```
ARGBLK: .BLKWO 10.            ;DEFINE DATA BLOCK
BUF:    .ASCII /TIMESUP/      ;DEFINE USER DATA
        .
        .
        .
        .NTIN  #ARGBLK,#15.,#0,#BUF,#7.,#0
```

The example below shows the *dollar* form of the same call.

```
ARGBLK: $NTIN  15.,0,BUF,7.
BUF:    .ASCII /TIMESUP/
        .
        .
        .
        MOV    #ARGBLK,R0
        JSR    PC,$$$MSR
```

5.6.7 Send Link Service Message (NTLS) (DECnet)

The NTLS call can be used in three ways: (1) to request data from a remote program over a flow-controlled logical link (Chapter 4); (2) to reenable incoming Interrupt Messages over a logical link (Chapter 4); or (3) to obtain status information for the link. No user data is allowed with a Link Service Message.

Macro Format:

.NTLS *area,ula,lsflgs,drcnt,ircnt*

or

label: $NTLS *ula,lsflgs,drcnt,ircnt*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTLS call, the argument block will contain the following:

ula	-7
////////////////	lsflgs
////////////////////////////////	
////////////////////////////////	
////////////////////////////////	
ircnt	drcnt
////////////////////////////////	
////////////////////////////////	
////////////////////////////////	
////////////////////////////////	

ula

The *ula* is a byte value specifying the user link address (1 to 255 decimal) chosen by the calling program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

lsflgs

The low-order two bits of this byte indicate the purpose of the call.

Bit 1	Bit 0	
0	0	Indicates the call is requesting segments or logical messages from the remote program. In this case, the *drcnt* argument is interpreted as the number of segments or logical messages being requested. This form of the call can be used only when the calling program requested message or segment flow control in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message for this logical link. If the program requested no flow control, an NTLS call with the low-order two bits of *lsflgs* = 00 will result in an error (BADFUO).
0	1	Indicates the call is to reenable incoming Interrupt Messages for this link. In this case the *ircnt* argument is interpreted as the number of interrupts being requested (always 1).
1	0	Indicates the call is to obtain status information only. In this case, the *drcnt* and *ircnt* arguments are ignored.

The sign bit of this byte is significant in the first two cases above. When the sign bit (bit 7) is set to 1, link status information will be returned to the FIRQB when the NTLS completes successfully.

Link status information is always returned when bit 1 is set. If it is requested, the information is returned to the FIRQB in the same format as a received Link Status Message (see Section 5.7.7).

A useful way to document and set these flags is to assign mnemonics to values that set (or clear) the appropriate bits. For example:

```
REQMSG  = 0
REQINT  = 1
RLS     = 128.
```

Then, to request a data message and return link status information, for example, the *lsflgs* argument would be specified as #REQMSG+RLS.

drcnt

The data request count byte (*drcnt*) is meaningful only when bits 0 and 1 of *lsflgs* are both equal to zero and indicates the maximum number of segments or logical messages being requested from the remote program. Depending on the flow control option in effect for the logical link, the value is interpreted as either a segment counter (segment flow control) or a logical message counter (message flow control). In either case it is an incremental count, added to the counter of outstanding requests for segments or logical messages kept by the remote and local NSP (see Chapter 4). An error is returned if the *drcnt* given increases the counter to more than 127 (decimal).

ircnt

The interrupt request count byte (*ircnt*) is meaningful only when the low-order two bits of *lsflgs* = 01 and indicates the number of Interrupt Messages being requested. DECnet/E allows only one interrupt request outstanding for a logical link at any given time. Hence, *ircnt* must always be 1 when the low-order two bits of *lsflgs* = 01 and has the effect of reenabling Interrupt Messages on the logical link. Remember that Interrupt Messages can be reenabled only when the previous Interrupt Message for this logical link, if any, has been received from the pending message queue. If it has not, the call will fail with an error (INTLCK) in R0.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
4	NOROOM	The Interrupt/Link Service transmit queue for this logical link is full. NSP is waiting for acknowledgment from the remote system for previously sent Interrupt or Link Service Messages. No Interrupt or Link Service Messages can be sent until at least one of the outstanding messages in the transmit queue is acknowledged. This condition is temporary and will not cause a Link Service Message to be queued when the condition clears. The program should simply retry the send after a short (1 to 5 second) delay. (See Chapter 4 for a full discussion of flow control.)
5	NOSUCH	The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending a Link Service Message over a logical link that has not yet been confirmed will cause this error.
18	BADFUO	One of several possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The low-order two bits of the *lsflgs* byte are both set to 1. 3. The low-order two bits of the *lsflgs* byte are 00 but the calling program did not request flow control for the logical link in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message. 4. The data request count byte (*drcnt*) increased the count of outstanding requests to more than 127 (decimal). 5. The low-order two bits of the *lsflgs* byte = 10, indicating a request for an Interrupt Message, but the *ircnt* argument is not equal to 1.

(continued on next page)

Decimal Value	ERR.STB Mnemonic	Meaning
		6. The low-order two bits of the *lsflgs* byte = 10, indicating a request for an Interrupt Message, but the outstanding requests counter for interrupts is already equal to one.
		7. The *ula* specified is zero. It must be within the range 1 to 255 (decimal).
19	INTLCK	The low-order two bits of *lsflgs* = 10, indicating a request for an Interrupt Message, but there is already an Interrupt Message queued for this logical link. See Chapter 4 for details on flow control.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example of NTLS requests five segments from the remote program over logical link 23. (Assume that the calling program requested segment flow control in its NTCI or NTCC call for this link.)

```
ARGBLK: .BLKW0 10.         ;DEFINE ARGUMENT BLOCK
        .
        .
        .
        .NTLS   #ARGBLK,#23.,#0,#5,#0
```

The following example shows the *dollar* form of the same call.

```
ARGBLK: $NTLS   23.,0,5
        .
        .
        .
        MOV     #ARGBLK,R0
        JSR     PC,$$$MSR
```

5.6.8 Send Disconnect Message (NTDI) (DECnet)

The NTDI call terminates an established logical link. Messages in the pending message queue will remain but no more messages can be sent over the link. The user link address is freed and can be used for another logical link. If the last Network Data Message sent did not have the end of message flag (bit 1 of the *dmflgs* argument) set, or if any Network Data, Interrupt, or Link Service Messages are still waiting for acknowledgment in the transmit queues for this logical link, this call will fail with an error.

Successful completion of a NTDI call assures the calling program that the remote system has received and acknowledged all Network Data, Interrupt, and Link Service Messages previously sent over the link. It does not, however, guarantee that the receiving program has processed the messages. The Disconnect Message is useful in "master-slave" communication where the master program only transmits data and the slave program only receives data (see Section 3.13). Otherwise, the NTDI call provides no particular advantage over a NTLA call (Section 5.6.9) in terminating a logical link. Up to 16 (decimal) bytes of user data can be sent with the Disconnect Message.

Macro Format:

```
        .NTDI area,ula,bufptr,buflen,bufoff
or
 label: $NTDI ula,bufptr,buflen,bufoff
```

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTDI call, the argument block will contain the following:

<table>
<tr><td align="center">ula</td><td align="center">-10</td></tr>
<tr><td colspan="2">/ /</td></tr>
<tr><td colspan="2" align="center">bufptr</td></tr>
<tr><td colspan="2" align="center">buflen</td></tr>
<tr><td colspan="2" align="center">bufoff</td></tr>
<tr><td colspan="2">/ /</td></tr>
<tr><td colspan="2">/ /</td></tr>
<tr><td colspan="2">/ /</td></tr>
<tr><td colspan="2">/ /</td></tr>
<tr><td colspan="2">/ /</td></tr>
</table>

ula

The *ula* is a byte value specifying the user link address (1 to 255 decimal) chosen by this program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

bufptr

This word defines the starting address of the optional user message data. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent: 0 to 16 (decimal).

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100 would be sent.

Possible Errors (Returned In R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
4	NOROOM	There are outstanding unacknowledged messages on either the Data Message transmit queue or the Interrupt/Link Service transmit queue. All messages previously sent over the logical link must be acknowledged before the Disconnect Message can be sent. (If an immediate unconditional termination of the logical link is desired, use the NTLA call.)
5	NOSUCH	The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
10	PRVIOL	Some procedural error has occurred. Sending a Disconnect Message over a logical link that has not yet been confirmed will cause this error.
18	BADFUO	One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The *ula* specified is zero. It must be within the range 1 to 255 (decimal).
19	INTLCK	The last segment sent over this logical link did not have the *eom* flag set. NSP will not permit the link to be disconnected unless all messages have been completely transmitted. (Note that this test for the end of message flag is done, irregardless of the flow control option chosen by the remote program.)
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Disconnect Message.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example disconnects logical link 255 and includes user data explaining the reason for the disconnect. (The remote program would have to be coded to recognize this message and take appropriate action.)

```
ARGBLK: .BLKWO 10.        ;DEFINE ARGUMENT BLOCK
MSGBUF: .ASCII /ALLDONE/
        .
        .
        .
        .NTDI  #ARGBLK,#255.,#MSGBUF,#7,#0
```

The following example shows the *dollar* form of the same call.

```
ARGBLK: $NTDI  255.,MSGBUF,7
MSGBUF: .ASCII /ALLDONE/
          .
          .
          .
        MOV    #ARGBLK,R0
        JSR    PC,$$$MSR
```

5.6.9 Send Link Abort Message (NTLA) (DECnet)

The NTLA call terminates a logical link immediately. If there are still messages in the transmit queues waiting for acknowledgment from the remote NSP, they will be discarded. They may or may not have been received by the remote system (see Section 3.14).

Up to 16 (decimal) bytes of user data can be sent with the message. If the NTLA call is issued for an established logical link, the remote system is notified that the link has been broken and the user data is delivered to the remote program. If the NTLA call is issued for a logical link awaiting confirmation from the remote program, the user data does not reach the remote program (see Section 3.14).

Macro Format:

.NTLA *area,ula,bufptr,buflen,bufoff*

or

label: $NTLA *ula,bufptr,buflen,bufoff*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

> Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the NTLA call, the argument block will contain the following:

ula	-11
/ /	
bufptr	
buflen	
bufoff	
/ /	
/ /	
/ /	
/ /	
/ /	

ula

The *ula* is a byte value specifying the user link address (1 to 255 decimal) chosen by this program in its Connect Initiate (NTCI) or Connect Confirm (NTCC) Message sent during the connection sequence.

bufptr

This word defines the starting address where the optional user message data begins. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the number of bytes of user data to be sent: 0 to 16 (decimal).

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 16 and a *bufoff* of -100. Sixteen bytes beginning at *bufptr*-100 would be sent.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
1	BADDIR	Network operations have been terminated due to an internal error. Notify the system manager.
5	NOSUCH	The user link address (*ula*) specified in the call does not correspond to any known logical link for the calling program. Either the *ula* is incorrect or the logical link has been disconnected. In the latter case, a Disconnect or Link Abort Message has already been queued for this link.
18	BADFUO	One of two possibilities: 1. The calling program is not a declared receiver. A Declare Receiver call must be issued before this call is given. 2. The *ula* specified is zero. It must be within the range 1 to 255 (decimal).
31	BADCNT	The length field passed in *buflen* is invalid. The length of the buffer must be between 0 and 16 for a Link Abort Message.
32	NOBUFS	System buffers are not currently available to store this message. A later try may succeed.
62	NORTS	NSP has not been enabled. Normal DECnet calls cannot be executed until the system manager enables NSP.
66	ERRERR	DECnet was not installed at system generation time. The network functions cannot be executed.

Example:

The following example aborts logical link 17. No user data is sent so the buffer arguments are set to zero.

```
ARGBLK: .BLKW0 10.        ;DEFINE ARGUMENT BLOCK
         .
         .
         .
        .NTLA   #ARGBLK,#17.,#0,#0,#0
```

The following example shows the *dollar* form of the same call.

```
ARGBLK: $NTLA   17.
         .
         .
         .
        MOV     #ARGBLK,R0
        JSR     PC,$$$MSR
```

The MRCV call retrieves a message from the calling program's pending message queue. It returns control parameters to the FIRQB and XRB, and user data, if any, to a buffer defined in the call. The Connect Data Block for Connect Initiate Messages is also delivered to the buffer. The user data can be retrieved all at once or in portions, and portions can be discarded. The formats in Sections 5.7.1 – 5.7.9 show what is returned to the FIRQB and XRB for the various types of messages.

The Receive call can be selected to get (1) the first message in the queue, (2) the first message in the queue from either a network or a local sender, or (3) the first message in the queue for a particular logical link. A "sleep" flag and associated timer can be set to indicate that the calling program is to wait if there are no appropriate messages pending in the queue.

Macro Format:

.MRCV *area,rmod,sndr,qual,bufptr,buflen,bufoff,slptim*

or

label: $MRCV *rmod,sndr,qual,bufptr,buflen,bufoff,slptim*

Argument Descriptions:

label

The *label* used for the *dollar* form of the call defines the starting address of the 10-word argument block created by the *dollar* form. The format of the argument block is the same as that described for *area*. The values are created at assembly time for the *dollar* form.

NOTE

Remember from the discussion in Section 5.1 that the *dot* form fills in only the arguments specified in the call. If an argument is omitted, the corresponding field in the argument block is left alone. The *dollar* form fills in zeros.

area

This word is the address of a 10-word argument block allocated in a data section of the program. After execution of the MRCV call, the argument block will contain the following:

/ / / / / / / / / / / / / /	2
sndr	rmod
bufptr	
buflen	
bufoff	
slptim	
/ / / / / / / / / / / / / /	qual
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /
/ / / / / / / / / / / / / /	/ / / / / / / / / / / / / /

rmod

The low-order four bits of this byte define the way the Receive call is executed.

bit 0 = 1 The program is to be put into a sleep state if no appropriate message is pending in the queue. The *slptime* parameter regulates the time that the program will sleep. The conditions under which the program is awakened are discussed under the *slptime* parameter.

= 0 The program should not be put into a sleep state if there is no appropriate pending message in the queue. The call terminates immediately with a NOSUCH error to inform the program that no appropriate messages are pending.

bit 1 = 1 Any excess data that will not fit in the specified buffer is to be discarded. Data could be left over if the value of *buflen* is smaller than the amount of user data in the message.

= 0 Excess data that does not fit in the buffer specified in this call is to be kept for retrieval on later Receive calls.

bit 2 = 1 A message is to be selected from a local sender. The selection can be further qualified with the *sndr* and *qual* arguments.

= 0 Local selection is not requested. If both bit 2 and bit 3 are zero, the first message in the queue is returned.

bit 3 = 1 A message is to be selected from a network sender. The selection can be further qualified with the *sndr* and *qual* arguments.

= 0 Network selection is not requested. If both bit 3 and bit 2 are zero, the first message in the queue is returned. If both bit 3 and bit 2 are one, local selection prevails.

One way to document and set these flags is to assign mnemonics to the values that will set the bits. For example:

```
SLP    = 1
TRUNC  = 2
LOCAL  = 4
NET    = 8.
```

To request a network message and the sleep option, then, the *rmod* argument could be coded as #NET+SLP.

sndr

The sender selection byte (*sndr*) selects a particular local sender or a particular logical link for this Receive call.

If bit 2 of *rmod* = 1 (local selection) and *sndr* is nonzero, *sndr* is interpreted as a job number times two. The first message on the queue from that particular local job is retrieved.

If bit 2 of *rmod* = 0 and bit 3 = 1, a nonzero value for *sndr* is interpreted as a user link address. The first message on the queue from that logical link is delivered.

If *sndr* = 0 when either bit 2 or bit 3 of *rmod* = 1, the *qual* argument has special meaning. The *sndr* and *qual* arguments are ignored if both bit 2 and bit 3 of *rmod* are zero.

qual

The sender selection qualifier (*qual*) byte is normally zero for user applications.

If bit 2 of *rmod* = 1 (local selection) and *sndr* = 0, then any nonzero value for *qual* is a special-case Receive call requesting a message from the system (job

number 0). This special case is used by the RSTS/E utility ERRCPY, which processes messages from the monitor error logging routines.

The *qual* argument is ignored if both bit 2 and bit 3 of *rmod* = 0 or if *sndr* ≠ 0.

Table 5–4 summarizes the relationships between the selection bits of *rmod* and the *sndr* and *qual* arguments.

Table 5–4: Sender Selection Summary

rmod bit 3	bit 2	*sndr*	*qual*	Result
0	0	n/a	n/a	The *sndr* and *qual* values are ignored. The MRCV call returns the first message in the pending message queue.
n/a	1	0	0	Selects the first Local Data Message in the queue.
		0	nonzero	Selects job 0. This combination is used by the error logging programs to select messages from monitor error logging routines.
		nonzero	n/a	Selects Local Data Messages by job number. The *sndr* argument is interpreted as a job number times 2. Only messages from that job are delivered on this MRCV call.
1	0	0	n/a	Selects the first network message (anything other than a Local Data Message).
		nonzero	n/a	Selects network messages from a particular logical link. The *sndr* argument is interpreted as a user link address. Only network messages from the designated logical link are delivered on this MRCV call.

bufptr

This word defines the starting address where the user message data is to be delivered. If a nonzero offset is given (see *bufoff*), the offset is added to the starting address defined by *bufptr*.

buflen

This word defines the maximum number of bytes to be delivered to the buffer on this Receive call.

If a program has limited buffer space, the *buflen* value can be set to process a large segment in small pieces. If truncation is not requested (bit 1 of *rmod* = 0) and the amount of data in the message is greater than the amount indicated by *buflen*, then only the amount indicated by *buflen* will be returned. The rest will be saved for retrieval with later Receive calls.

The usual sequence in this case is to issue Receive calls until the number of bytes remaining in the pending message goes to zero. This point can be determined by examining the word at FIRQB+12 in the data returned by the Receive call. The number of bytes actually returned to the buffer is returned in the word at XRB+2. (See the formats in Sections 5.7.1 – 5.7.9.)

bufoff

This word defines an offset, in bytes, from the address specified by the *bufptr* argument. The value of *bufoff* is added to the *bufptr* address to define the beginning address of the user message data.

NOTE

> The MACRO interface makes no checks for inconsistencies in the *bufptr*, *buflen*, and *bufoff* values specified. A program could, for example, specify a *buflen* of 100 and a *bufoff* of 20. Up to 100 bytes beginning at *bufptr*+20 could be delivered on the Receive call.

slptim

The *slptime* argument is significant only when bit 0 of *rmod* = 1. In that case, *slptime* defines the length of time, in seconds, that a program is to sleep when no appropriate message is in the queue. The program will sleep until:

1. The sleep timer (*slptime*) expires.
2. *Any* new message is placed in the queue (not just an appropriate one).
3. A delimiter is typed on a terminal opened by the job or assigned to the job.
4. Log-ins are disabled. (This could occur if the system is being shut down).
5. A state change occurs on a pseudo-keyboard assigned to the job. (The job has printed output for the controlling job to read, or has entered an input wait state.)
6. The job itself has opened the XM: (DMC11 driver) and a message is received for the job.

The program is awakened in all cases with an error (NOSUCH) but is not passed as a message. To obtain a pending message the program must execute another Receive call. Since the program can be awakened by any of the conditions listed, a check for pending messages can be made by executing a MRCV without a sleep.

Possible Errors (Returned in R0):

Decimal Value	ERR.STB Mnemonic	Meaning
5	NOSUCH	For a Receive call without sleep (bit 0 in *rmod* = 0), this error indicates that no appropriate messages are pending. For a Receive call with sleep (bit 0 in *rmod* = 1), this error is returned when the program is awakened from the sleep. The program must execute another MRCV call to retrieve any pending messages.
18	BADFUO	No Receiver ID block exists for this program. Before any Receive call can succeed, a Declare Receiver (MDCL) call must be executed.

Example:

In the following example a Receive call is issued for network messages from logical link 25. The sleep bit is set and the timer is set to five seconds. The truncate bit is 0 so all the information that is available to be returned to the buffer is saved for another Receive call. (The program will examine the byte at FIRQB+4 to see what type of message is available and will jump to an appropriate section of code depending upon its value.)

```
ARGBLK: .BLKWO 10.          ;DEFINE ARGUMENT BLOCK
         .
         .
         .
        .MRCV   #ARGBLK,#9.,#25.,#0,,#0,#0,#5
```

The following example shows the *dollar* form of the same call:

```
ARGBLK: $MRCV   9.,25.,0,0,0,5
         .
         .
         .
        MOV     #ARGBLK,R0
        JSR     PC,$$$MSR
```

5.7.1 Format of Received Local Data Message

The format of the information returned to the FIRQB and XRB for a Local Data Message is shown below. Up to 512 (decimal) bytes of user data can be delivered to the buffer specified in the Receive (MRCV) call.

FIRQB

Octal Offset			Octal Offset
1	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	0
3	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	2
5	job number * 2	-1 (function code)	4
7	project number	programmer number	6
11	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	10
13	remainder (number of bytes not transferred)		12
15	parameters		14
17	(Data passed as parameters by the sender of this message, padded with zeros to 20 bytes)		16
21			20
23			22
25			24
27			26
31			30
33			32
35			34
37			36

FIRQB+4 The function code for a Local Data Message.

FIRQB+5 Two times the job number of the local sender.

FIRQB+6 The project-programmer number of the local sender.

FIRQB+12 The number of bytes of user data not delivered to the buffer. This data has been discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument in the MRCV call.

FIRQB+14 The data passed as parameters by the sender of this message is returned starting at this address. The system pads any unused bytes with zeros to a length of 20 bytes.

XRB

Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 — The length (bytes) of information transferred to the buffer on this Receive call.

5.7.2 Format of Received Connect Initiate Message

The format of the information returned to the FIRQB and XRB for a received Connect Initiate Message is shown below. A 120 (decimal) byte Connect Data Block and up to 16 (decimal) bytes of user data are also returned to the buffer specified in the Receive call. The format of a received Connect Data Block is shown in Figure 5–2 and described in detail in Table 5–5.

FIRQB

Octal Offset			Octal Offset
1	//////////	//////////	0
3	//////////	//////////	2
5	//////////	-2 (function code)	4
7	lla (local link address)		6
11	//////////	//////////	10
13	remainder (number of bytes not transferred)		12
15	//////////	//////////	14
17	//////////	rlmod: 0=no, 1=sg, 2=ms	16
21	rmax (maximum bytes received in 1 network data message)		20
23	tmax (maximum bytes sent in 1 network data message)		22
25	//////////	//////////	24
27	//////////	//////////	26
31	//////////	//////////	30
33	//////////	//////////	32
35	//////////	//////////	34
37	//////////	//////////	36

FIRQB+4 The function code for a Connect Initiate Message.

FIRQB+6 Local link address. Identifies the link for the receiving program's subsequent Connect Confirm or Connect Reject Message.

FIRQB+10 The number of bytes of data not returned to the buffer. This data has been discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument in the Receive call.

FIRQB+16 Remote link modifier. Indicates the flow control in effect for the remote program:

0 = No flow control.
1 = Segment flow control.
2 = Message flow control.

FIRQB+20 Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the receiving program in one Network Data Message over this link. This value is supplied by the local NSP and is determined by the size of the receive buffers that it allocates. The maximum size of the receive buffers is set for NSP by network management. The receiving program can further limit the amount of user data to be received by specifying a smaller *rmax* argument in the Connect Confirm Message that accepts this request for a connection. (See Section 5.6.3).

FIRQB+22 Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the receiving program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote NSP. The local NSP enforces this limit for all Network Data Messages sent by the local program on this link.

XRB

Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

Decimal Position	Octal Offset			Octal Offset	Decimal Position
		REMOTE NODE NAME IN ASCII (SPACE FILL TO 6 BYTES)		0	1
6	5			4	5
8	7	OBJECT TYPE	FORMAT	6	7
10	11	DESCRIPTOR LENGTH		10	9
12	13	REMOTE DESCRIPTOR		12	11
⋮	⋮				
22	25			24	21
24	27	GROUP CODE FOR FORMAT 2		26	23
26	31	USER CODE FOR FORMAT 2		30	25
28	33	OBJECT TYPE	FORMAT	32	27
30	35	DESCRIPTOR LENGTH		34	29
		LOCAL DESCRIPTOR		36	31
⋮	⋮			⋮	⋮
44	53	GROUP CODE FOR FORMAT 2		52	43
46	55	USER CODE FOR FORMAT 2		54	45
48	57	ID LENGTH		56	47
50	61	USER IDENTIFICATION		60	49
⋮	⋮			⋮	⋮
64	77			76	63
66	101	PASSWORD LENGTH		100	65
68	103	PASSWORD		102	67
74	111			110	73
76	113	ACCOUNT LENGTH		112	75
78	115	ACCOUNTING INFORMATION		114	77
92	133			132	91
94	135	NOT MEANINGFUL — SHOULD BE IGNORED		134	93
120	167			166	119

Figure 5–2: Format of Received Connect Data Block

Table 5–5: Format of Connect Data Block on Received Connect Initiate Message

Octal Offset	Content
0–5	**Remote Node.** 1 to 6 uppercase, alphanumeric ASCII characters naming the remote node from which the Connect Initiate Message was received. The name will contain at least one alphabetic character. Node names shorter than six characters will be left-justified and padded to six characters with spaces.
6–31	**Remote Program.** Identifies the remote program that sent the Connect Initiate Message as either an object type code (general function) or a specific entity (program, task, process). This identification is given in one of three formats:

Format 0

The remote program is identified by object type code alone. The fields are:

Octal Offset	Content
6	Zero, to indicate a format 0 name.
7	Object type code of the remote program.
10–31	Zero (Not used.)

Format 1

The remote program is identified by a descriptor. The fields are:

Octal Offset	Content
6	One, to indicate a format 1 name.
7	Normally zero. This byte will usually be zero unless the remote DECnet system allows identification by both name and object type code on outgoing connect requests (DECnet/E does not). If nonzero, this byte is interpreted as the object type code of the remote program that sent the Connect Initiate Message.
10	Reserved - must be zero.
11	Length of descriptor, 0 to 16 (decimal) bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (offset 12–31 octal).
12–31	Remote program descriptor (left-justified). The descriptor used to register the remote program for network operations. For remote DECnet/E systems this is the logical name declared in the Declare Receiver call.

Format 2

This format identifies the remote program by name or object type code and by the group and user codes under which the remote program is running on the remote system. (The group and user codes refer to what is called the project-programmer number on some DIGITAL systems, including RSTS/E. Other systems refer to these codes as the user identification code, or UIC.) DECnet/E systems use this format to identify the sender of a Connect Initiate Message. The subfields are defined as follows:

Octal Offset	Content
6	Two, to indicate a format 2 name.
7	Object type code of the remote program, if applicable. If zero, the descriptor alone identifies the remote program.
10	Reserved - must be zero.
11	Length of descriptor, 0 to 12 (decimal) bytes. Gives the number of characters to be interpreted as the program descriptor in the following field (octal offset 12–25).

(continued on next page)

Table 5-5 (Cont.): Format of Connect Data Block on Received Connect Initiate Messsage

Octal Offset	Content
12–25	Remote program descriptor, if this is used instead of object type code. The descriptor will be left-justified in the field.
26–27	Remote group code. Binary value giving the group number under which the remote program is running. If the remote system is a DECnet/E system, the byte at octal offset 26 is the project number in a project-programmer number.
30–31	Remote user code. Binary value giving the user number under which the remote program is running. If the remote system is a DECnet/E system, the byte at octal offset 30 is the programmer number in a project-programmer number.

32–55 **Local Program.** This field indicates how the remote program addressed the local program in the Connect Initiate Message. DECnet/E allows the local program to be addressed by object type code or by name, but not both. Again, three formats can be used. The subfields are as follows:

Octal Offset	Content
32	Format type. Equal to 0, 1, or 2, depending on the format used by the remote program to address this one.
33	Object type code, if used by the remote program to address the local program (formats 0 or 2).
34	Always zero.
35	Length of name, in bytes. Will be 0 for format 0, 1 to 6 for format 1, 0 to 6 for format 2. (DECnet/E program names are limited to a maximum of six characters by the Declare Receiver call.)
36–43	Local program name, if used by the remote program to address the local program (formats 1 or 2).
44–51	Always zero.
52–53	Local group code. Binary value giving the local program's project number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by NSP to identify the local program but passed on here.
54–55	Local user code. Binary value giving the local program's programmer number, if used by the remote program to address the local program (format 2). Zero otherwise (formats 0 or 1). Not used by NSP to identify the local program but passed on here.

56–133 **Access Control Fields.** These bytes can be used to define the remote program's access rights to the local program's services. The idea is analogous to logging in to the local system. The most rigorous checking done in a normal log-in on DIGITAL systems involve user identification, password, and account number so three fields are defined here for those items. However, overall DECnet design does not specifically require use of these fields at all, nor does it require that the fields be used as described. A DECnet/E system simply passes this information on to the receiving progr_m.

(continued on next page)

Table 5-5 (Cont.): Format of Connect Data Block on Received Connect Initiate Messsage

Octal Offset	Content
56	Not meaningful - should be ignored.
57	Length of user identification information specified in the next field, 0 to 16 (decimal) bytes.
60-77	User Identification. This field identifies the person or program requesting the logical link. In RSTS/E terms, the user ID is analogous to the project-programmer number used at log-in.
100	Not meaningful - should be ignored.
101	Length of password information specified in the next field, 0 to 8 (decimal) bytes.
102-121	Password. A password is often used to verify access to a system under a particular user identification. In RSTS/E, this password is analogous to the password used in conjunction with the project-programmer number used at log-in.
122	Not meaningful - should be ignored.
123	Length of the accounting information specified in the following field, 0 to 16 (decimal) bytes.
124-143	Accounting information. Many systems require a billing account number in addition to user identification and password. The accounting field is provided for this purpose.

134-167	Not meaningful - should be ignored.

5.7.3 Format of Received Connect Confirm Message

A received Connect Confirm Message is a remote program's acceptance of a Connect Initiate Message sent by the local program. The link is identified by the user link address specified by the local program in the Connect Initiate Message. The format of the information returned to the FIRQB and XRB is described below. Up to 16 bytes of user data can also be returned to the buffer specified in the Receive call.

FIRQB

Octal Offset			Octal Offset
1	///////////////	///////////////	0
3	///////////////	///////////////	2
5	ula (user link address)	-3 (function code)	4
7	///////////////	///////////////	6
11	///////////////	///////////////	10
13	remainder (number of bytes not transferred)		12
15	///////////////	///////////////	14
17	///////////////	rlmod: 0=no, 1=sg, 2=ms	16
21	rmax (maximum bytes received in 1 network data message)		20
23	tmax (maximum bytes sent in 1 network data message)		22
25	///////////////	///////////////	24
27	///////////////	///////////////	26
31	///////////////	///////////////	30
33	///////////////	///////////////	32
35	///////////////	///////////////	34
37	///////////////	///////////////	36

FIRQB+4 The function code for a Connect Confirm Message.

FIRQB+5 User link address, previously defined in the receiving program's Connect Initiate Message.

FIRQB+12 Number of bytes of data not returned to the buffer. This data has been discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument for this Receive call.

FIRQB+16 Remote link modifier. Indicates the flow control in effect for the remote program:

0 = No flow control.
1 = Segment flow control.
2 = Message flow control.

FIRQB+20 Receive maximum, in bytes. The maximum amount of user data that will be transmitted to the local program in one Network Data Message over this link. This will be (1) equal to the *rmax* value specified by the local program in the Connect Initiate Message for this link, or (2) a lesser value supplied by the local NSP, determined from the maximum size of the receive buffers it allocates.

FIRQB+22 Transmit maximum, in bytes. The maximum amount of user data that can be transmitted by the local program in one Network Data Message over this link. This limit has been imposed by either the remote program or the remote NSP. The local NSP enforces this limit for all Network Data Messages sent by the local program on this link.

XRB

Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

5.7.4 Format of Received Connect Reject Message

A received Connect Reject Message is a rejection of the receiving program's request for a logical link. It can be from the remote program or from the local or remote NSP. The link is identified by the user link address previously established in the receiving program's Connect Initiate Message. The format of the information returned to the FIRQB and XRB is described below. Rejections from NSP will contain a reason code explaining the reason for the rejection and will contain no user data. Rejections from the remote program can contain up to 16 (decimal) bytes of user data delivered to the buffer defined in the Receive call.

FIRQB

Octal Offset			Octal Offset
1	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	0
3	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	2
5	ula (user link address)	-4 (function code)	4
7	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	6
11	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	10
13	remainder (number of bytes not transferred)		12
15	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	14
17	reason (for rejection)		16
21	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	20
23	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	22
25	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	24
27	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	26
31	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	30
33	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	32
35	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	34
37	/ / / / / / / / / / / / / / / /	/ / / / / / / / / / / / / / / /	36

FIRQB+4 Function code for Connect Reject Message.

FIRQB+5 User link address, previously defined in the receiving program's Connect Initiate Message.

FIRQB+12 Number of bytes of data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument on the Receive call.

FIRQB+16 Identifies the reason for the rejection (see Appendix B).

XRB

Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

5.7.5 Format of Received Network Data Message

A received Network Data Message contains user data from a remote program. The format of the information returned to the FIRQB and XRB is described below. The user data accompanying the Network Data Message is delivered to the buffer specified in the Receive call.

FIRQB

Octal Offset			Octal Offset
1	//////////////////	//////////////////	0
3	//////////////////	//////////////////	2
5	ula (user link address)	-5 (function code)	4
7	//////////////////	//////////////////	6
11	//////////////////	//////////////////	10
13	remainder (number of bytes not transferred)		12
15	//////////////////	dmflgs	14
17	//////////////////	//////////////////	16
21	//////////////////	//////////////////	20
23	//////////////////	//////////////////	22
25	//////////////////	//////////////////	24
27	//////////////////	//////////////////	26
31	//////////////////	//////////////////	30
33	//////////////////	//////////////////	32
35	//////////////////	//////////////////	34
37	//////////////////	//////////////////	36

FIRQB+4 Function code of Network Data Message.

FIRQB+5 User link address, previously defined in this program's Connect Initiate or Connect Confirm Message.

FIRQB+12 Number of bytes of data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument on the Receive call.

FIRQB+14 Data message flags. The low-order two bits of this byte indicate whether this is the beginning, middle, end, or sole segment of a logical message.

Bit 1	Bit 0	
0	0	Middle segment of a message.
0	1	First segment of a message.
1	0	Last segment of a message.
1	1	Sole segment of a message.

If this program requested message flow control when the logical link was established, DECnet/E uses bit 1 to determine the end of a logical message. It does not use bit 0 but passes it on to the receiving program.

XRB

Octal Offset		Octal Offset
1	//////////////////////////////	0
3	length (actual number of bytes transferred)	2
5	//////////////////////////////	4
7	//////////////////////////////	6
11	//////////////////////////////	10
13	//////////////////////////////	12
15	//////////////////////////////	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

5.7.6 Format of Received Interrupt Message

The local NSP places Interrupt Messages from a remote program at the head of the pending message queue, behind the first message and behind any other pending Interrupt Messages queued for the program. The format of the data returned to the FIRQB and XRB is described below. Up to 16 (decimal) bytes of user data can be delivered to the buffer specified in the Receive call.

FIRQB

Octal Offset			Octal Offset
1	/ /		0
3	/ /		2
5	ula (user link address)	-6 (function code)	4
7	/ /		6
11	/ /		10
13	remainder (number of bytes not transferred)		12
15	/ /		14
17	/ /		16
21	/ /		20
23	/ /		22
25	/ /		24
27	/ /		26
31	/ /		30
33	/ /		32
35	/ /		34
37	/ /		36

FIRQB+4 — Function code for an Interrupt Message.

FIRQB+5 — User link address, previously defined by this program's Connect Initiate or Connect Confirm Message.

FIRQB+12 — The number of bytes of data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate bit was set in the *rmod* argument on this Receive call.

XRB

Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

5.7.7 Format of Received Link Service Message

A Link Service Message provides status information for a particular logical link. A Link Service Message is only returned on a Receive call when the pending message queue has emptied after a condition has cleared that previously inhibited this program from sending (see Section 4.2). Status information (in the same format as a Link Service Message) can also be returned to the FIRQB and XRB on successful completion of a Send Link Service, Send Network Data, or Send Interrupt call. The format of the information returned is described below. No user data is delivered to the buffer with a Link Service Message.

FIRQB

Octal Offset			Octal Offset
1	//////////////////////////////////////		0
3	//////////////////////////////////////		2
5	ula (user link address)	-7 (function code)	4
7	//////////////////////////////////////		6
11	//////////////////////////////////////		10
13	(always = 0)		12
15	rstat (remote status)	lstat (local status)	14
17	lir (local interrupt cnt)	ldr (local data count)	16
21	rir (remote interrupt cnt)	rdr (remote data count)	20
23	dtc (data xmit que cnt)	dtm (data xmit que max)	22
25	ltc (lnk srvc xmit que cnt)	ltm (lnk srvc xmit que max)	24
27	mcnt (message count)	mmax (message max)	26
31	/////////////////	mula (messages for link)	30
33	//////////////////////////////////////		32
35	//////////////////////////////////////		34
37	//////////////////////////////////////		36

FIRQB+4	Function code for Link Service Message.
FIRQB+5	User link address, previously defined by this program's Connect Initiate or Connect Confirm Message.
FIRQB+12	Always zero for Link Service Messages. (This word is the remainder field in other returned messages.)

FIRQB+14	Local status flags. The two low-order bits give the status of outgoing messages:

Bit 0 = 1 Outgoing Network Data Messages are inhibited. A Link Service Message will be delivered when the condition clears.

= 0 No restrictions on sending Network Data Messages.

Bit 1 = 1 Outgoing Interrupt and Link Service Messages are inhibited. A Link Service Message will be delivered when the condition clears.

The two high-order bits give the status of incoming messages:

Bit 7	Bit 6	
0	0	Incoming Data Message flow is on.
0	1	(Decimal value = 64) Incoming Data Message flow is on but the local NSP will turn this link off if another Data Message is received before this program's pending message queue is emptied.
1	0	(Decimal value = 128) Incoming Data Message flow has been turned off by the local NSP. This program's pending message queue must be emptied before the link will be turned on.
1	1	(Decimal value = 192) Incoming Data Message flow has been turned off by the local NSP but flow is scheduled to turn on as soon as system buffers become available.

FIRQB+15 Remote status flag.

Bit 7 = 1 (Decimal value = 128) The link has been turned off by the remote system due to a backpressure condition. No Network Data Messages can be sent until the remote system turns the link back on.

FIRQB+16 Local data request count: The count of segments or logical messages currently requested by this program, reflecting the actual number of segments or logical messages requested but not yet received.

FIRQB+17 Local interrupt request count: The count of Interrupt Messages currently requested by this program, reflecting the actual count of interrupts requested but not yet received. This count will never exceed one.

FIRQB+20 Remote data request count: The count of segments or logical messages currently requestd by the remote program, reflecting the actual count of segments or logical messages requested but not yet sent.

FIRQB+21 Remote interrupt request count: The count of Interrupt Messages currently requested by the remote end of the logical link, reflecting the actual count of interrupts requested but not yet sent.

FIRQB+22 Data transmit queue maximum: The maximum number of data messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when NSP is enabled. It applies individually to all active logical links.

FIRQB+23 Data transmit queue count: The number of data messages currently in the data transmit queue for this logical link. It is a count of data messages waiting for acknowledgment from the remote NSP.

FIRQB+24 Interrupt/Link Service transmit queue maximum: The maximum number of Interrupt or Link Service Messages that can be queued waiting for acknowledgment from the remote system. This is a constant set by the system manager when NSP is enabled. It applies individually to all active logical links.

FIRQB+25 — Interrupt/Link Service transmit queue count: The number of Interrupt and Link Service Messages currently in the Interrupt/Link Service transmit queue for this logical link. It is a count of Interrupt and Link Service Messages waiting for acknowledgment from the remote NSP.

FIRQB+26 — Message maximum: The maximum number of incoming messages that will be queued for this program. This maximum is set by this program in the Declare Receiver call.

FIRQB+27 — Message count: The total number of messages currently in the pending message queue for this program.

FIRQB+30 — Message count for this logical link: The number of messages currently in the pending message queue for this logical link, as identified by the user link address (FIRQB+5).

XRB

Octal Offset		Octal Offset
1	/ /	0
3	(always = 0)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 — Always zero for Link Service Messages. (This word is the length field in other returned messages.)

5.7.8 Format of Received Disconnect Message

A received Disconnect Message indicates that an established logical link with a remote program has been terminated by the remote program. All other messages sent by the remote program over this logical link have been received by this program. The format of the data returned to the FIRQB and XRB is defined below. Up to 16 (decimal) bytes of user message data can be returned to the buffer specified in the Receive call.

FIRQB

Octal Offset			Octal Offset
1	//////////////////	//////////////////	0
3	//////////////////	//////////////////	2
5	ula (user link address)	-10 octal (function code)	4
7	//////////////////	//////////////////	6
11	//////////////////	//////////////////	10
13	remainder (number of bytes not transferred)		12
15	//////////////////	//////////////////	14
17	//////////////////	//////////////////	16
21	//////////////////	//////////////////	20
23	//////////////////	//////////////////	22
25	//////////////////	//////////////////	24
27	//////////////////	//////////////////	26
31	//////////////////	//////////////////	30
33	//////////////////	//////////////////	32
35	//////////////////	//////////////////	34
37	//////////////////	//////////////////	36

FIRQB+4 Function code for a Disconnect Message.

FIRQB+5 User link address, previously defined in this program's Connect Initiate or Connect Confirm Message.

FIRQB+12 Number of bytes of user message data not returned to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate flag was set in the *rmod* argument for the Receive call.

XRB

Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

5.7.9 Format of Received Link Abort Message

A received Link Abort Message indicates that a logical link has been terminated by the remote program or by the local or remote NSP. The format of the data returned to the FIRQB and XRB is described below. Up to 16 (decimal) bytes of user message data can be returned to the buffer specified in the Receive call.

FIRQB

Octal Offset			Octal Offset
1	//////////////////////////////	//////////////////////////////	0
3	//////////////////////////////	//////////////////////////////	2
5	ula (user link address)	-11 octal (function code)	4
7	lla (local link address)		6
11	//////////////////////////////	//////////////////////////////	10
13	remainder (number of bytes not transferred)		12
15	//////////////////////////////	//////////////////////////////	14
17	reason (for link abort)		16
21	//////////////////////////////	//////////////////////////////	20
23	//////////////////////////////	//////////////////////////////	22
25	//////////////////////////////	//////////////////////////////	24
27	//////////////////////////////	//////////////////////////////	26
31	//////////////////////////////	//////////////////////////////	30
33	//////////////////////////////	//////////////////////////////	32
35	//////////////////////////////	//////////////////////////////	34
37	//////////////////////////////	//////////////////////////////	36

FIRQB+4 Function code for Link Abort Message.

FIRQB+5 User link address, if any, previously defined in the receiving program's Connect Initiate or Connect Confirm Message.

FIRQB+6 Local link address established by local NSP for this link. Identifies the link if the remote program sent a Connect Initiate Message but the local program has not yet responded with a Connect Confirm or Connect Reject Message. No user link address exists under these conditions to identify the link being aborted.

FIRQB+12 Number of bytes of user data not transferred to the buffer. This data has been either discarded or saved for a later Receive call, depending on how the truncate flag was set in the *rmod* argument for the Receive call.

FIRQB+16 Abort reason. If this value is nonzero, the link was aborted by either the local or remote NSP and the reason is specified in this word. The reasons that apply for a link abort are listed in Appendix B. If this word is zero, the link was aborted at the request of the remote program.

XRB

Octal Offset		Octal Offset
1	/ /	0
3	length (actual number of bytes transferred)	2
5	/ /	4
7	/ /	6
11	/ /	10
13	/ /	12
15	/ /	14

XRB+2 The length (bytes) of information transferred to the buffer on this Receive call.

Appendix A
Object Type Codes

Code	Type of Process
000	General Task, User Process
001	File Access (FAL/DAP Version 1)
002	Unit Record Services (URD)
003	Application Terminal Services (ATS)
004	Command Terminal Services (CTS)
005	RSX-11M Task Control, Version 1
006	Operator Services Interface
007	Node Resource Manager
008	IBM 3270-BSC Gateway
009	IBM 2780-BSC Gateway
010	IBM 3790-SDLC Gateway
011	TPS Application
012	RT-11 DIBOL Application
013	TOPS-20 Terminal Handler
014	TOPS-20 Remote Spooler
015	RSX-11M Task Control, Version 2
016	TLK Utility (LSN on DECnet/E)
017	File Access (FAL/DAP Version 4 and later)
018	RSX-11S Remote Task Loader
019	Network Management Listener (NICE Process)
020	RSTS/E Media Transfer Program (NETCPY)
021	Reserved for DECnet use
022	Mail Listener
023	Host Terminal Handler (NPKDVR)
024	Concentrator Terminal Handler

025	Loopback Mirror
026	Event Receiver
027	VAX/VMS Personal Message Utility
028	File Transfer Spooler (FTS)
029–062	Reserved for DECnet use
063	DECnet test tool (DTR)
064–127	Reserved for DECnet use
128–255	Reserved for customer extensions

Appendix B
NSP Reason Codes for Connect Reject and Link Abort

Code	Reason
000	No error – user-initiated reject or abort
001	Resource allocation failure
002	Destination node does not exist
003	Node shutting down
004	Destination program does not exist
005	Invalid destination name or source name field
006	Destination program's queue full
007	Unspecified error condition
008	Third party aborted logical link
009	User-initiated link abort
010–020	Reserved
021	Invalid destination address in Connect Initiate Message
022	Invalid destination address in Connect Confirm Message
023	Source address zero in Connect Initiate or Connect Confirm Message
024	Flow control violation – invalid request count in Link Service Message
025–031	Reserved
032	Too many connects to node
033	Too many connects to destination program
034	Access not permitted
035	Logical link services mismatch
036	Invalid accounting information
037	Segment size too small
038	User aborted, timed out, or canceled link

039	No path to node
040	Flow control failure – data received when request count zero
041	No current link (cannot recognize destination address)
042	Confirmation from remote system of Disconnect Message
043	Image data field too long

Appendix C
FIRQB and XRB Layouts for Message Calls

When one of the message macro calls, described in Chapter 5, is used in a MACRO program, subroutine $$$MSR is invoked. This subroutine first sets up the job's File Request Queue Block (FIRQB) and Transmit Request Block (XRB), using parameters passed to it by the macro call. (The FIRQB and XRB are two areas within the first 1000 bytes of virtual address space reserved for communications between the RSTS/E monitor, the run-time system, and the user job. A general description of the first 1000 bytes of virtual memory, along with the FIRQB and the XRB, can be found in the *RSTS/E System Directives Manual.*) It then executes a .MESAG monitor directive. This directive transfers program control to the appropriate section of the NSP code, residing within the RSTS/E monitor, which then performs the function requested.

This appendix contains the FIRQB and XRB layouts for each of the message calls. These layouts are provided for reference purposes only. You should *not* attempt to set up the FIRQB and the XRB and invoke the .MESAG directive yourself. DECnet send and receive operations should be accessed *only* with the macro calls described in Chapter 5.

In the following sections, the parameters passed are not described in detail. They do, however, carry the same mnemonic abbreviations as their corresponding macro parameters described in Chapter 5. For a detailed explanation of these parameters, refer to that chapter.

C.1 Declare Receive

FIRQB

Octal Offset			Octal Offset
1	////////////////////////////////		0
3	////////////////////////////////		2
5	////////////////	1 (function code)	4
7	name (receiver logical name in ASCII, space fill to six bytes)		6
11			10
13			12
15	access code	objtyp (object type)	14
17	bmax (buffer maximum)		16
21	lmax (link maximum)	mmax (message maximum)	20
23	////////////////////////////////		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Declare Receiver subfunction of .MESAG.

C.2 Remove Receiver

FIRQB

Octal Offset			Octal Offset
1	/ /		0
3	/ /		2
5	0 or job number*2	0 (function code)	4
7	(must be = 0)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)		14
17	(must be = 0)		16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Remove Receiver subfunction of .MESAG.

C.3 Get Local Node Parameters

FIRQB

Octal Offset			Octal Offset
1	/ /		0
3	/ /		2
5	1	-19 (function code)	4
7	(must be = 0)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)		14
17	(must be = 0)		16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)		32
35	(must be = 0)		34
37	(must be = 0)		36

Data Returned:

In addition to a status code in byte 0, the following data is returned in the FIRQB:

FIRQB

Octal Offset		Octal Offset
1	//////////////////////////////	0
3	//////////////////////////////	2
5	//////////////////////////////	4
7	local node name	6
11	(left-justified and space filled)	10
13		12
15	local node address	14
17	local node alias	16
21	(left-justified and space filled)	20
23		22
25	default project-programmer number (0 if none defined)	24
27	//////////////////////////////	26
31	//////////////////////////////	30
33	//////////////////////////////	32
35	//////////////////////////////	34
37	//////////////////////////////	36

C.4 Log User Event

FIRQB

Octal Offset			Octal Offset
1			0
3			2
5	evmod (error modifier)	-10 (function code)	4
7	event code (bits 0–4 = evtyp; bits 6–14 = evcls)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	enval (entity value or 0 if no entity)		14
17	(must be = 0)	entyp (entity type)	16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)		32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of output buffer, in bytes (0 - 512)	0
3	number of bytes to send (0 - buffer length)	2
5	starting address of buffer	4
7		6
11	(must be = 0)	10
13	(must be = 0)	12
15		14

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned for the Log User Event subfunction of .MESAG.

C.5 Send Local Data Message

FIRQB

Octal Offset			Octal Offset
1	/ /		0
3	/ /		2
5	0 or job number*2	-1 (function code)	4
7	name		6
11	(receiver logical name in ASCII, space fill to six bytes)		10
13			12
15	param		14
17	(optional user parameter string: up to 20 bytes of additional user data can be specified here, zero fill to 20 bytes)		16
21			20
23			22
25			24
27			26
31			30
33			32
35			34
37			36

XRB

Octal Offset		Octal Offset
1	length of output buffer, in bytes (0 - 512)	0
3	number of bytes to send (0 - buffer length)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Local Data Message subfunction of .MESAG.

C.6 Send Connect Initiate Message

FIRQB

Octal Offset			Octal Offset
1	/ /		0
3	/ /		2
5	ula (user link address)	-2 (function code)	4
7	(must be = 0)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)		14
17	(must be = 0)	llmod (link modifier)	16
21	rmax (maximum bytes in received network data message)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (120 - 136)	0
3	number of bytes to send (0 - buffer length)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned for the Send Connect Initiate subfunction of .MESAG.

C.7 Send Connect Confirm Message

FIRQB

Octal Offset			Octal Offset
1			0
3			2
5	ula (user link address)	-3 (function code)	4
7	lla (local link address)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)		14
17	(must be = 0)	llmod (link modifier)	16
21	rmax (maximum bytes in received network data message)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (0 if no user data)	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7		6
11	(must be = 0)	10
13	(must be = 0)	12
15		14

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Connect Confirm subfunction of .MESAG.

C.8 Send Connect Reject Message

FIRQB

Octal Offset			Octal Offset
1	/ /		0
3	/ /		2
5	/ / / / / / / / / / / / / / / /	–4 (function code)	4
7	lla (local link address)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)		14
17	reason code		16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (0 if no user data)	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7	/ /	6
11	(must be = 0)	10
13	(must be = 0)	12
15	/ /	14

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Connect Reject subfunction of .MESAG.

C.9 Send Network Data Message

FIRQB

Octal Offset			Octal Offset
1			0
3			2
5	ula (user link address)	-5 (function code)	4
7	(must be = 0)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)	dmflgs (data flags)	14
17	(must be = 0)		16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Octal Offset		Offset
1	length of buffer, in bytes	0
3	number of bytes to send (0 - 532)	2
5	starting address of buffer	4
7		6
11	(must be = 0)	10
13	(must be = 0)	12
15		14

Data Returned:

If bit 7 is set in the *dmflgs* byte (FIRQB+14) of the input parameters, data is returned in the FIRQB in the same format as a received Link Service Message (see Section 5.7.7).

C.10 Send Interrupt Message

FIRQB

Octal Offset			Octal Offset
1			0
3			2
5	ula (user link address)	-6 (function code)	4
7	(must be = 0)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)	inflgs (interrupt flags)	14
17	(must be = 0)		16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7		6
11	(must be = 0)	10
13	(must be = 0)	12
15		14

Data Returned:

If bit 7 is set in the *inflgs* byte (FIRQB+14) of the input parameters, data is returned in the FIRQB in the same format as a received Link Service Message (see Section 5.7.7).

C.11 Send Link Service Message

FIRQB

Octal Offset			Octal Offset
1			0
3			2
5	ula (user link address)	-7 (function code)	4
7	(must be = 0)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)	lsflgs (link service flags)	14
17	lrcnt (interrupt req count)	drcnt (data req count)	16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (0 if no user data)	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7		6
11	(must be = 0)	10
13	(must be = 0)	12
15		14

Data Returned:

If bit 7 is set in the *lsflgs* byte (FIRQB+14) of the input parameters, data is returned in the FIRQB in the same format as a received Link Service Message (see Section 5.7.7).

C.12 Send Disconnect Message

FIRQB

Octal Offset			Octal Offset
1	//////////////////////////////		0
3	//////////////////////////////		2
5	ula (user link address)	-8 (function code)	4
7	//////////////////////////////		6
11	//////////////////////////////		10
13	//////////////////////////////		12
15	//////////////////////////////		14
17	//////////////////////////////		16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (0 if no user data)	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7	//////////////////////////////	6
11	(must be = 0)	10
13	(must be = 0)	12
15	//////////////////////////////	14

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Disconnect subfunction of .MESAG.

C.13 Send Link Abort Message

FIRQB

Octal Offset			Octal Offset
1	//////////////////////////////		0
3	//////////////////////////////		2
5	ula (user link address)	-9 (function code)	4
7	(must be = 0)		6
11	(must be = 0)		10
13	(must be = 0)		12
15	(must be = 0)		14
17	(must be = 0)		16
21	(must be = 0)		20
23	(must be = 0)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (0 if no user data)	0
3	number of bytes to send (0 - 16)	2
5	starting address of buffer	4
7	//////////////////////////////	6
11	(must be = 0)	10
13	(must be = 0)	12
15	//////////////////////////////	14

Data Returned:

Except for a possible error code in byte 0 of the FIRQB, no data is returned by the Send Link Abort subfunction of .MESAG.

C.14 Receive Message

FIRQB

Octal Offset			Octal Offset
1	//////////////////////////////		0
3	//////////////////////////////		2
5	rmod (receive modifiers)	2 (function code)	4
7	qual (normally 0)	sndr (sender select)	6
11	(must be = 0)		10
13	(must be = 0)		12
15	//////////////////////////////		14
17	//////////////////////////////		16
21	//////////////////////////////		20
23	slptime (sleep time, in seconds)		22
25	(must be = 0)		24
27	(must be = 0)		26
31	(must be = 0)		30
33	(must be = 0)	RIB number	32
35	(must be = 0)		34
37	(must be = 0)		36

XRB

Octal Offset		Octal Offset
1	length of buffer, in bytes (0 if no data desired)	0
3	(must be = 0)	2
5	starting address of buffer	4
7	//////////////////////////////	6
11	(must be = 0)	10
13	(must be = 0)	12
15	//////////////////////////////	14

Data Returned:

The Receive call returns data identifying the type of message received to the FIRQB and XRB, and user data (if any) to the buffer defined at XRB+4.

The FIRQB and XRB formats for the various message types are shown in Sections 5.7.1 – 5.7.9.

INDEX

M

N

O

P

Q

R

S

T

U

V

X

DECnet/E
Network Programming
in MACRO-11
AA-L265A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)__________

Name__________ Date__________

Organization__________

Street__________

City__________ State__________ Zip Code__________
or
Country

—Do Not Tear - Fold Here and Tape —

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE DOCUMENTATION
1925 ANDOVER STREET TW/E07
TEWKSBURY, MASSACHUSETTS 01876

— Do Not Tear - Fold Here and Tape —

Cut Along Dotted Line